Modélisation intégratrice du traitement BigData

Hadi Hashem

NNT : 2016SACLL005

THESE DE DOCTORAT
DE
L'UNIVERSITE PARIS-SACLAY
PREPAREE A
TELECOM SUDPARIS

ÉCOLE DOCTORALE STIC
Sciences et technologies de l'information et de la communication

Spécialité de doctorat : réseaux, information et communications

Par

Mr Hadi Hashem

Modélisation intégratrice du traitement BigData

Thèse présentée et soutenue à Évry, le 19 septembre 2016 :

Composition du Jury :

Mme. K. Zeitouni, Professeure, Université de Versailles Saint-Quentin, Présidente du jury
Mme. M. Sibilla, Professeure, Université de Toulouse 3 - Paul Sabatier, Rapporteur
M. L. D'Orazio, Professeur, Université Blaise-Pascal, Rapporteur
Mme N. Simoni, Professeure, Télécom ParisTech, Examinatrice
Mme. G. Vargas-Solar, Chercheuse, CNRS, Examinatrice
M. F. Masseglia, Chercheur, INRIA, Examinateur
Mme. A.R. Cavalli, Professeure, Télécom SudParis, Directrice de thèse
M. D. Ranc, Enseignant-Chercheur, Télécom SudParis, Co-encadrant

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Modélisation intégratrice du traitement BigData

Mots clés : modélisation intégratrice, BigData, raisonnement à base de cas

Résumé : Dans le monde d'aujourd'hui de multiples acteurs de la technologie numérique produisent des quantités infinies de données. Capteurs, réseaux sociaux ou e-commerce, ils génèrent tous de l'information qui s'incrémente en temps-réel selon les 3 V de Gartner : en Volume, en Vitesse et en Variabilité. Afin d'exploiter efficacement et durablement ces données, il est important de respecter la dynamicité de leur évolution chronologique au moyen de deux approches : le polymorphisme d'une part, au moyen d'un modèle dynamique capable de supporter le changement de type à chaque instant sans failles de traitement ; d'autre part le support de la volatilité par un modèle intelligent prenant en compte des données clé seulement interprétables à un instant « t », au lieu de traiter toute la volumétrie des données actuelle et historique.
L'objectif premier de cette étude est de pouvoir établir au moyen de ces approches une vision intégratrice du cycle de vie des données qui s'établit selon 3 étapes, (1) la synthèse des données via la sélection des valeurs-clés des micro-données acquises par les différents opérateurs au niveau de la source, (2) la fusion en faisant le tri des valeurs-clés sélectionnées et les dupliquant suivant un aspect de dé-normalisation afin d'obtenir un traitement plus rapide des données et (3) la transformation en un format particulier de carte de cartes de cartes, via Hadoop dans le processus classique de MapReduce afin d'obtenir un graphe défini dans la couche applicative.
Cette réflexion est en outre soutenue par un prototype logiciel mettant en œuvre les opérateurs de modélisation sus-décrits et aboutissant à une boîte à outils de modélisation comparable à un AGL et, permettant une mise en place assistée d'un ou plusieurs traitements sur BigData.

Title : Integrative modeling of Big Data processing

Keywords : integrative modeling, BigData, case-based reasonning

Abstract : Nowadays, multiple actors of Internet technology are producing very large amounts of data. Sensors, social media or e-commerce, all generate real-time extending information based on the 3 Vs of Gartner: Volume, Velocity and Variety. In order to efficiently exploit this data, it is important to keep track of the dynamic aspect of their chronological evolution by means of two main approaches: the polymorphism, a dynamic model able to support type changes every second with a successful processing and second, the support of data volatility by means of an intelligent model taking in consideration key-data, salient and valuable at a specific moment without processing all volumes of history and up to date data.
The primary goal of this study is to establish, based on these approaches, an integrative vision of data life cycle set on 3 steps, (1) data synthesis by selecting key-values of micro-data acquired by different data source operators, (2) data fusion by sorting and duplicating the selected key-values based on a de-normalization aspect in order to get a faster processing of data and (3) the data transformation into a specific format of map of maps of maps, via Hadoop in the standard MapReduce process, in order to define the related graph in applicative layer.
In addition, this study is supported by a software prototype using the already described modeling tools, as a toolbox compared to an automatic programming software and allowing to create a customized processing chain of BigData.

Remerciements

Je tiens à remercier Madame Ana Rosa CAVALLI, directrice de thèse et Monsieur Daniel RANC, co-encadrant, de m'avoir soutenu durant toute la période de préparation de mon travail.

Je remercie la présidente du jury, madame Karine ZEITOUNI, Professeure de l'Université de Versailles Saint-Quentin à Vélizy, ainsi que les membres du jury :

1- Madame Michelle SIBILLA, Professeure de l'Université Toulouse 3 - Paul Sabatier à Toulouse.
2- Monsieur Laurent D'ORAZIO, Professeur de l'Université Blaise-Pascal à Clermont-Ferrand.
3- Madame Noémie SIMONI, Professeure de Télécom ParisTech à Paris.
4- Madame Genoveva VARGAS-SOLAR, Chercheuse au Centre National de Recherche Scientifique à Saint Martin d'Hères.
5- Monsieur Florent MASSEGLIA, Chercheur à l'Institut National de Recherche en Informatique et en Automatique à Montpellier.

Je les remercie de leurs directives, de leurs précieux conseils et d'avoir donné l'attention nécessaire pour évaluer mon travail.

J'adresse mes remerciements à Télécom SudParis et particulièrement au Département Réseaux et Services Multimédia Mobiles au sein duquel j'ai pu développer ce travail, discuter et échanger avec ses membres. Enfin, je remercie ma petite famille et toutes les personnes formidables autour de moi qui m'ont encouragé et supporté physiquement et moralement pour mener à bien mon travail.

Table des matières

Table des illustrations

Les figures

Les tableaux

Introduction de la thèse

1. La problématique et le contexte du travail

Des volumes considérables de données sont créés tous les jours à partir des données utilisateur générées automatiquement sur Internet. Réseaux sociaux, appareils mobiles, messagerie électronique, blogs, vidéos, transactions bancaires et autres interactions utilisateur pilotent désormais les campagnes Marketing, les études sociodémographiques, les enquêtes de polices et les intentions électorales, en établissant une nouvelle dimension appelée BigData.

Les moteurs de base de données basés sur le standard SQL et créés dans les années 1970 ont de bonnes performances lors du traitement de petites quantités de données relationnelles mais ces outils sont très limités face à l'expansion des données en volume et en complexité. Le traitement MPP créé initialement au début des années 1980 a amélioré légèrement les indicateurs de performance pour les volumes de données complexes. Cependant, ce traitement n'a pas pu être utilisé pour le traitement des données non-relationnelles à expansion permanente.

Des outils puissants sont requis pour stocker et exploiter ces données en expansion quotidienne dans le but de soumettre un traitement simple et fiable des données récoltées des utilisateurs. Des résultats rapides et de bonne qualité sont attendus. Pour les industriels et les décideurs en général, ces résultats sont aussi importants que les plus lourds investissements métier. Les opérateurs de modélisation traditionnels sont confrontés à leurs limitations dans ce défi, puisque les informations se multiplient en volume et complexité, une chose qui actuellement ne peut être gérée que par des techniques de modélisation non-relationnelles. Hadoop MapReduce est considéré comme la technique de traitement la plus efficace, comparée aux bases de données SQL et au traitement MPP. Hadoop dispose d'une performance proportionnelle à la complexité des données volumineuses. C'est un outil efficace pour résoudre les problèmes de données massives mais c'est aussi un concept qui a changé l'organisation des systèmes de traitement en large échelle. Cependant malgré le succès qu'il a eu, ce modèle n'a pas encore atteint son aspect final en tant que solution informatique mature. Au contraire, il s'agit d'un point de départ vers d'autres perspectives.

Par ailleurs, l'interaction consommateur sur Internet est considérée comme un nouveau canal digital entre les marques et leur audience. Plusieurs EOs de données sont créés au quotidien sous forme d'information basée sur des modèles de données en expansion continue, en volume et complexité. Les modèles de notation consommateur intégrant des fonctionnalités de prédiction ont atteint des résultats significatifs en termes de taux de conversion. En utilisant des techniques statistiques et d'autres données consommateurs disponibles sur Internet, des modèles de prédiction personnalisés peuvent être créés afin d'identifier le potentiel des consommateurs.

Dans le contexte de notre recherche, le travail consiste à adresser cette question en se basant sur une boîte à outils contenant des opérateurs de modélisation permettant d'établir un pré-traitement des données avant l'envoi au serveur de calcul. Ce travail propose également un mariage de deux technologies du domaine informatique pour créer un modèle d'application générique : les systèmes de gestion des bases de données (SGBD) et le raisonnement par étude de cas (CBR). Les SGBD fournissent des facilités bas-niveau, en revanche, ils assurent une assistance minime en termes d'interface utilisateur et d'extraction de données. Les SGBD ne permettent pas de faire des raisonnements logiques à partir des données stockées, ce qui empêche de mettre en avant la valeur intrinsèque des données. Comme nous le verrons, le SGBD couplé à un moteur d'inférence CBR est plus performant et plus efficace sur cet aspect.

Le rapprochement entre ces deux techniques permet d'obtenir un concept personnalisable, facilitant la création d'une chaîne de traitement basée sur des opérateurs de modélisation à la carte et profitant des performances de calcul de Hadoop MapReduce. Il s'agit donc d'un traitement BigData en utilisant les règles du raisonnement par étude de cas à l'échelle des réseaux distribuées et garantissant un traitement décentralisé, séquentiel, isolé du développeur et évolutif selon le besoin en vigueur.

2. Les objectifs de la thèse

L'objectif premier de cette étude est de contribuer à l'établissement d'une vision intégratrice du cycle de vie des données. Cette vision s'intéresse en particulier, mais sans exclusive, au pré-traitement des données et s'appuie sur les trois étapes suivantes :

1- L'acquisition des micro-données diverses et variées, de sources multiples, de tailles, de sémantiques et de formats différents, à travers des connecteurs assurant une conversion des flux en fichiers à stocker, selon le modèle de base de données utilisé.
2- Le pré-traitement via des opérateurs de modélisation sélectionnés par l'utilisateur selon une configuration précise et adéquate avec son besoin, dans le but d'identifier les données nécessaires pour calculer le résultat final parmi le reste.
3- Le traitement des données présélectionnées par les opérateurs de modélisation dans le moteur de calcul et l'obtention d'une indication sur le résultat final recherché.

Cette vision intégratrice mènera à l'étude d'un modèle de pré-traitement à base de cas reposant sur un rapprochement entre un système expert et un système de gestion de base de données permettant d'élaborer un concept de moteur d'inférence avec une base de connaissance de prédicats. Ce modèle étant un moyen efficace pour lancer un pré-traitement des données BigData en se basant sur des cas similaires et permettant par conséquent d'arriver rapidement à une indication sur le résultat final, avec un niveau de tolérance raisonnable. Les approches proposées ont été validées par des prototypes logiciels traitant des jeux de données réalistes et exhibant des gains d'efficacité tangibles.

Enfin, dans le cadre de cette étude, on veille à proposer un modèle intuitif clé-en-main permettant d'améliorer les performances du traitement avec des coûts moins importants, ne nécessitant pas une connaissance technique approfondie dans un domaine technologique en expansion continue et ayant à la fois un impact positif sur les performances de la chaîne de traitement, par conséquent, sur l'environnement.

3. Le plan de la thèse

Ce travail est organisé en 3 parties.

La première partie introduira l'état de l'art du traitement des données BigData.

Dans le premier chapitre, on détaille l'évolution des systèmes de gestion des bases de données non-relationnelles. La base de données NoSQL sera introduite, dont l'usage est le plus répandu aujourd'hui dans les technologies de traitement BigData. Ensuite on introduit sa dérivée, la base de données NewSQL, tout en exposant son architecture, ses avantages, ainsi que ses limitations. On exposera ensuite la technologie Hadoop MapReduce dans le cadre d'une analyse de l'efficacité des moteurs de traitement existant. Cela permettra de définir par la suite les différents modèles de base de données non-relationnels existants et leur usage, ainsi que les bases de données multi-modèle. Enfin on consacrera le dernier paragraphe à l'activité principale des systèmes distribués en termes de consistance, de création des données, de coordination, ainsi que les autres aspects de gestion notamment la répartition de la charge, la tolérance aux pannes et la haute disponibilité. On terminera par la description des difficultés générales de mise en œuvre.

Le deuxième chapitre introduira le Framework MapReduce et ses principales caractéristiques le mettant en avant par rapport aux autres technologies. On présentera par la suite les différentes techniques de traitement et patrons de conception, tels que le tri, les jointures, l'indexation, le classement et la conversion. Le traitement des graphes avec MapReduce sera abordé, ainsi que les algorithmes de traitement de texte. Ensuite on évoquera les différents projets et évolutions de l'univers MapReduce, en particulier la nouvelle génération appelée YARN et les principaux projets dérivés de Hadoop, Apache Storm et Apache Spark. Finalement ce chapitre fournira un tableau comparatif des différentes possibilités.

Le troisième chapitre exposera les recherches portant plus particulièrement sur l'approche de la modélisation intégratrice. On définira également les trois grandes familles des techniques de modélisation, la modélisation conceptuelle, la modélisation générale et la modélisation hiérarchique. Ensuite, on présentera le périmètre de cette recherche et la motivation de ce travail qui consiste à proposer un modèle de traitement intuitif et clé-en-main, ne nécessitant pas une connaissance technique approfondie dans le domaine BigData et permettant d'optimiser les performances de la chaîne de traitement.

Le quatrième chapitre explique en détail les principaux algorithmes de modélisation avec MapReduce. Cela comprend les principaux opérateurs de modélisation, tels que le filtre, le découpage, la transformation ou la fusion, ainsi que les patrons basiques et non-basiques de MapReduce. Dans cette catégorie, on définit les algorithmes d'agrégation et d'assemblage, le tri, les tâches distribuées, ainsi que les algorithmes de traitement des graphes. Par la suite, on évoque les patrons relationnels MapReduce, comme la sélection, l'intersection, la projection, l'union, les jointures et d'autres. Pour finir, on présente l'API Trident d'Apache Storm, ainsi que les potentiels de l'apprentissage automatique avec MapReduce.

La seconde partie décrira le travail élaboré pour approcher la problématique de la modélisation.

Le cinquième chapitre introduit un algorithme de pré-traitement via un raisonnement par étude de cas et ce en en deux parties. D'abord, on présente brièvement les systèmes experts et les avantages d'un rapprochement avec les systèmes de gestion des bases de données. On explique par la suite le concept du moteur d'inférence basé sur les règles et les profils à définir, ainsi que son utilisation dans un contexte de modélisation intégratrice à l'échelle BigData. Pour finir, on évoquera les perspectives d'enrichissement de la base de cas via l'apprentissage automatique. Dans la deuxième partie, on se situera dans un contexte de surveillance des réseaux sociaux. On appliquera alors le concept de pré-traitement par étude de cas et son adaptation aux besoins métier.

Le sixième chapitre permet de présenter quelques cas d'emploi, ainsi que les résultats expérimentaux. Dans ce contexte, on réalisera d'abord une étude des données Twitter suivie d'une autre étude plus globale, en utilisant un développement en mode agile créé à ce propos. Ensuite, on abordera le pré-traitement par étude de cas, à travers 3 cas d'emploi adaptés à la vie quotidienne en entreprise et le besoin d'outils performants de traitement des données :

1- L'évaluation du profil revendeur
2- Les changements dans le trafic routier
3- La détection d'un taux d'attrition élevé

La troisième partie présentera la conclusion et les perspectives de développement de ce travail dans la modélisation en général et dans le pré-traitement par étude de cas.

Partie 1. Etat de l'art

Chapitre 1. Le traitement des données BigData

1.1 Introduction au chapitre

Depuis leur création, les bases de données, à taille petite ou volumineuse, sont devenues une entité essentielle et inséparable d'un applicatif ou d'un site Internet quelconque. Les bases de données relationnelles les plus répandues à l'époque avaient leurs SGBD disponibles par défaut dans les systèmes informatiques. Avec l'expansion du nombre d'internautes et la multitude des terminaux et objets connectés, les bases de données relationnelles ne sont plus capables de supporter les données volumineuses à stocker, extraire, déplacer et copier, surtout si elles sont distribuées sur plusieurs serveurs. D'où la nécessité d'une nouvelle génération de bases de données avancées, compatible avec l'étendue géographique des réseaux immenses de serveurs, dits Clusters et capable de gérer des quantités importantes de données.

La Figure 1 montre l'évolution des systèmes d'information et des données échangées depuis la création des réseaux informatique jusqu'à la nouvelle génération du Web, permettant aux internautes de contribuer à l'échange d'information et d'interagir de façon simple, à la fois au niveau du contenu et de la structure des pages, créant notamment le Web social.

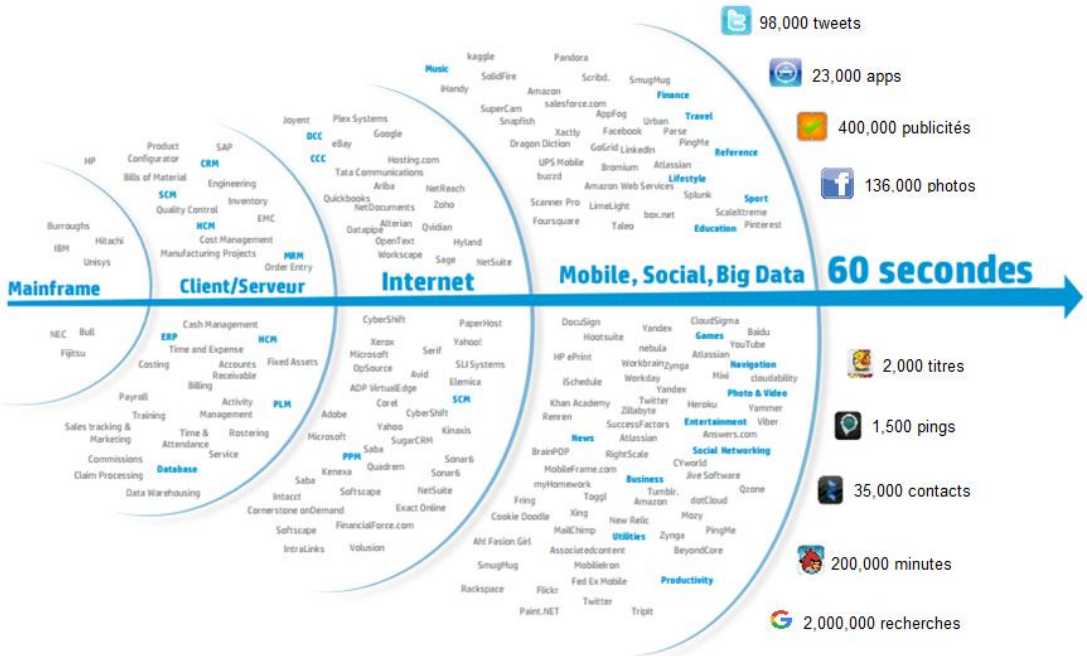

Figure 1 : Expansion exponentielle des données échangées sur Internet

1.2 Les bases de données NoSQL

De nos jours, l'ubiquité de la connexion Internet est une réalité (les voitures que nous conduisons, les montres que nous portons, nos petits appareils médicaux domestiques, nos réfrigérateurs et congélateurs, nos Smartphones et ordinateurs portables). De plus, les données numériques produites par les êtres humains, dont les séquences vidéo, les photos et autres, atteignent des volumes importants de plusieurs EO par jour. Ces données actuellement stockées dans des bases qui leur ont été conçues spécifiquement sont gérés par des logiciels de gestion de bases de données volumineuses, jouant le rôle d'intermédiaires entre les bases de données d'un côté et les applicatifs et leurs utilisateurs de l'autre. On parle ici des bases de données non-relationnelles, dites NoSQL.

1.2.1 Le mouvement NoSQL et l'élaboration du terme

Carlo Strozzi a utilisé le terme NoSQL ou Not Only SQL en premier en 1998 pour désigner la base de données relationnelle Open Source qu'il a développée et qui ne disposait pas d'une interface SQL comme ses homologues. Carlo Strozzi a proposé par la suite de changer le terme NoSQL en NoRel pour non-relationnelles, vu que ce mouvement a convergé avec le temps vers les bases de données non-relationnelles uniquement. En 2009, le terme NoSQL a été réintroduit par Eric Evans à une échelle plus large, décrivant les nombreuses bases de données s'opposant à la notion relationnelle et possédant les caractéristiques suivantes :

1- Elles sont toutes compatibles avec les systèmes distribués.
2- Elles sont de type Open Source.
3- Elles sont de type non-relationnel

1.2.2 La définition NoSQL et les avantages pour les développeurs

NoSQL est un type spécifique de bases de données, permettant de stocker et de récupérer les données après restructuration, en utilisant des techniques différentes de celles connues dans les bases de données relationnelles. Les développeurs de nos jours ont tendance à utiliser ce type de bases de données pour la simplicité de leur implémentation et leur évolutivité sans limite (horizontalement, à travers de nouvelles colonnes).

Afin d'obtenir de meilleures performances, les bases de données NoSQL ont abandonné certaines fonctionnalités proposées par défaut par les bases relationnelles comme les transactions ou encore les vérifications d'intégrités. Le premier besoin fondamental auquel répond NoSQL est la performance. C'est pour répondre à ce besoin que cette solution a vu le jour en procédant à des compromis sur le caractère ACID des systèmes de gestion de bases de données relationnels. Ces intelligents compromis sur la notion de relationnel ont permis de dégager les systèmes de gestion de bases de données relationnels de leurs freins à l'évolutivité.

De nos jours, NoSQL est devenu inséparable du BigData, le terme décrivant les données volumineuses et en expansion permanente, ainsi que des applicatifs temps-réel. Cette technologie remplace progressivement les bases de données relationnelles, leurs basses performances et leur coût élevé.

1.2.3 Les caractéristiques des bases de données NoSQL

Les bases de données NoSQL regroupent plusieurs caractéristiques apportant chacune une valeur ajoutée à leur usage :

1- Le coût raisonnable et la facilité de mise en œuvre.
2- Le partitionnement et la copie des fichiers de données sur plusieurs machines.
3- La structure dynamique n'ayant pas de schéma de données fixe.
4- L'évolutivité en rajoutant des colonnes, ce qui permet de traiter les données plus rapidement, surtout les plus volumineuses.
5- La rapidité du transfert des données comparé aux bases de données classiques.
6- L'évolutivité en rajoutant des nœuds supplémentaires dans le Cluster sans avoir besoin de faire une répartition.

De plus les bases de données NoSQL sont sujettes au théorème CAP et ne sont pas conformes aux propriétés ACID, contrairement aux bases de données relationnelles. Les réseaux sociaux appliquent fortement l'utilisation des bases de données NoSQL, vu leurs besoins compatibles avec CAP et contrairement aux banques nécessitant plus de rigidité.

1.2.3.1 Les propriétés ACID

Il s'agit d'un ensemble de propriétés qui garantissent une transaction exécutée de façon fiable :

1- L'atomicité, dite Atomicity, est une propriété qui assure qu'une transaction se fait au complet ou pas du tout. Si une partie d'une transaction ne peut être faite, il faut effacer toute trace de la transaction et remettre les données dans l'état où elles étaient avant la transaction. L'atomicité doit être respectée dans toute situation, comme une panne d'électricité, une défaillance de l'ordinateur ou une panne d'un disque magnétique.
2- La consistance, dite Consistency, qui assure que chaque transaction amènera le système d'un état valide à un autre état valide. Tout changement à la base de données doit être valide selon toutes les règles définies, incluant mais non limitées aux contraintes d'intégrité, aux Restaurations du système en cascade, dites Rollbacks, aux déclencheurs de base de données et à toute combinaison d'évènements.
3- L'isolation, dite Isolation, qui fait en sorte que toute transaction doit s'exécuter comme si elle était la seule sur le système. Aucune dépendance possible entre les transactions. Cette propriété assure que l'exécution simultanée de transactions produit le même état que celui qui serait obtenu par l'exécution en série des transactions. Chaque transaction doit s'exécuter en isolation totale. Si deux transactions s'exécutent simultanément, alors chacune doit demeurer indépendante de l'autre.
4- La durabilité, dite Durability, qui assure que lorsqu'une transaction a été confirmée, elle demeure enregistrée même à la suite d'une panne d'électricité, d'une panne de l'ordinateur ou d'un autre problème. Par exemple, dans une base de données relationnelle, lorsqu'un groupe de requêtes SQL ont été exécutés, les résultats doivent être enregistrés de façon permanente, même dans le cas d'une panne immédiatement après l'exécution des requêtes.

1.2.3.2 Le théorème CAP

Le théorème CAP, en français dit CDP, connu également sous le nom de théorème de Brewer, affirme qu'il est impossible sur un système informatique de calcul distribué de garantir en même temps les 3 contraintes de consistance, disponibilité et persistance au morcellement :

1- La consistance, dite Consistency, de façon à ce que tous les nœuds du système voient exactement les mêmes données au même moment.
2- La disponibilité, dite Availability, de façon à garantir que toutes les requêtes reçoivent une réponse.
3- La persistance au morcellement, dite Partition Tolerance, faisant en sorte qu'aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome).

1.2.3.3 La consistance des données

Mis à part leurs nombreux avantages, les bases de données NoSQL ne sont pas à l'abri des problèmes de consistance des données. Les développeurs des applications et les concepteurs de bases de données doivent gérer cet aspect selon la nature du métier. A titre d'exemple, sur un site Internet de réservation de chambres d'hôtels, il est possible que deux personnes puissent réserver à un intervalle de temps relativement réduit une même chambre d'hôtel. Il sera envoyé par la suite un mail à la personne qui a réservé en deuxième, lui expliquant que sa réservation n'a pas été prise en considération. Même principe lors de l'achat d'un produit sur une boutique en ligne. Les administrateurs des sites marchands préfèrent ce fonctionnement, plutôt que d'afficher un message d'erreur à l'écran, invitant l'utilisateur à recommencer.

1.2.4 Les limitations des bases de données NoSQL

Globalement les systèmes NoSQL ne respectent pas les propriétés ACID ou en tout cas pas complètement. Cet aspect ne permet pas d'offrir une grande sûreté dans l'accès aux données. Par ailleurs la base de données NoSQL reste très contraignante par certains aspects. Ainsi le traitement des requêtes de type OLAP nécessite une programmation importante au niveau applicatif.

1.2.5 Conclusion

Tout d'abord, cette génération de bases de données est encore relativement jeune et n'a pas encore atteint l'apogée de la maturité. Seules les petites et les moyennes entreprises les font évoluer pour le moment. Les grandes entreprises comme Oracle et IBM mettent encore en avant leurs SGBD classiques pour ce qu'elles offrent en termes de stabilité et de structuration.

1.3 NewSQL en route vers la base de données moderne

NewSQL est un stockage distribué et potentiellement entièrement en mémoire et pouvant être requêté classiquement par une interface SQL. NewSQL est tiré du monde NoSQL mais reste différent. Comme NoSQL il s'agit d'une nouvelle architecture logicielle qui propose de repenser le stockage des données. Elle profite des architectures distribuées, des progrès du matériel et des connaissances théoriques depuis 35 ans. Mais contrairement à NoSQL elle permet de conserver le modèle relationnel au cœur du système.

NewSQL est né de la rencontre de 3 types d'architecture, relationnelle, non-relationnelle et grille de données appelée également cache distribué, comme indiqué dans la Figure 2. En effet il se positionne comme un stockage distribué conçu dans le prolongement des architectures NoSQL, pour des accès transactionnels à fort débit, au moyen d'une interface SQL. D'un point de vue évolutivité, il se situe en tant que concurrent direct des solutions NoSQL. Mais contrairement à ces solutions il conserve une interface relationnelle via le SQL, ce qui est l'une de ses forces. Enfin la plupart des solutions NewSQL proposent un stockage en mémoire. Ce stockage en mémoire distribué sur plusieurs machines sous forme de grille de données est largement utilisé depuis une dizaine d'années dans les environnements où une faible latence est critique, notamment dans certaines applications des banques d'investissement. Les solutions NewSQL partagent ainsi un positionnement intermédiaire entre les solutions NoSQL et les grilles de données.

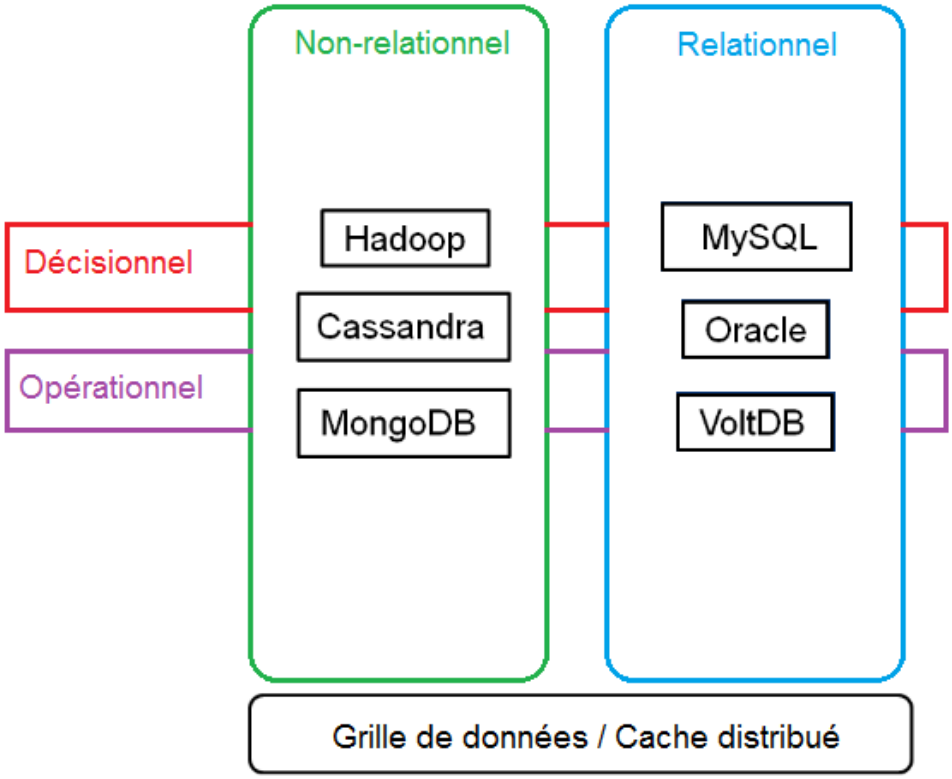

Figure 2 : Naissance du NewSQL à partir de 3 architectures

1.3.1 L'architecture NewSQL

L'architecture NewSQL reprend des expériences antérieures du SQL relationnel et du NoSQL plusieurs caractéristiques, tout en ayant certaines particularités en termes de choix et d'avantages :

1- Le choix d'une interface SQL et d'un schéma relationnel.
2- Le schéma relationnel avec des limitations pour faciliter la distribution des données et des traitements.
3- La distribution et la réplication des données pour assurer l'évolutivité et la résilience.

1.3.2 Les avantages de la solution NewSQL

La solution NewSQL présente des avantages intéressants en termes de performances par rapport à ses prédécesseurs :

1- Elle utilise le SQL comme langage commun de requêtes.
2- Elle présente une architecture qui a de meilleures performances par nœud que les solutions classiques de type SGBD relationnel.
3- Elle minimise la complexité des applications tout en améliorant la consistance des données et en fournissant un support transactionnel complet.
4- Elle est compatible avec les outils de travail du standard SQL.
5- Elle fournit des analyses plus riches des traitements.
6- Elle est compatible avec l'architecture distribuée des Clusters.
7- Elle fournit un traitement plus performant des données en mémoire.

1.2.3 Les limitations des bases de données NewSQL

La solution NewSQL est confrontée à quelques limitations l'empêchant d'atteindre un niveau de maturité suffisant :

1- Son architecture de calcul en mémoire, dit In-Memory, ne fonctionne pas au-delà de quelques TO de données.
2- Cette architecture nécessite un matériel spécifique avec des capacités importantes de stockage en mémoire, ce qui revient très onéreux.
3- Malgré son attitude à vouloir intégrer le modèle relationnel, le NewSQL fournit un accès limité aux outils de travail du standard SQL.

1.3.4 Conclusion

Une base de données NewSQL conserve la structure classique en colonnes mais fait appel à différents procédés pour conserver la rapidité même sur de larges volumes. En revanche, cette technologie n'a pas le recul suffisant pour aller plus loin et n'a pas encore pu suffisamment faire ses preuves. Par conséquent les entreprises sont encore réticentes à l'adoption de cette toute nouvelle architecture.

1.4 L'efficacité des moteurs de traitement existants

Hadoop MapReduce est considéré comme la technique de traitement la plus efficace, comparée aux bases de données SQL et au traitement MPP. Hadoop dispose de performances à l'échelle de la complexité des données volumineuses. La Figure 3 donne une indication sur la performance de traitement de Hadoop comparée aux deux autres moteurs de traitement.

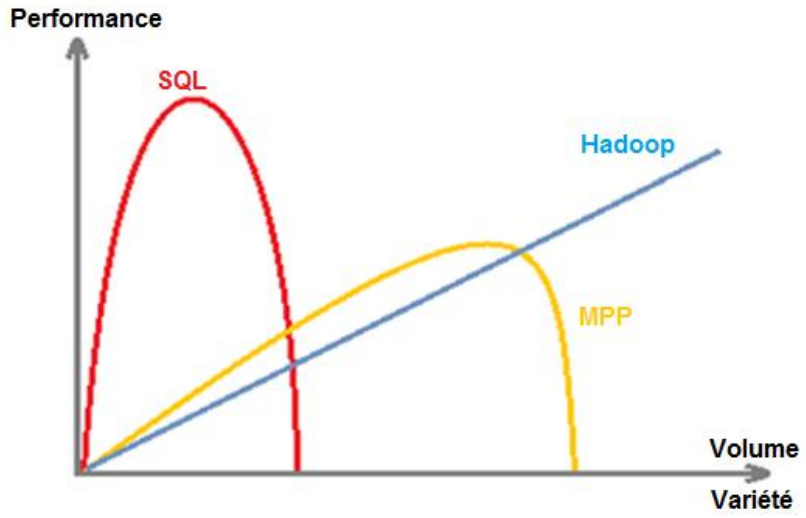

Figure 3 : Performance des moteurs de traitement

1.4.1 MapReduce

En 2004, Google a publié un nouvel article introduisant l'utilisation d'une technique simplifiée de traitement des données affichant de hautes performances dans le traitement des données volumineuses. Un modèle aussi intuitif que le MapReduce ne nécessite pas d'expertise concernant le parallélisme et les systèmes distribués. Son Framework Plug-and-Play embarque tous les détails pour implémenter les systèmes de calcul parallèle, la persistance et la résilience, l'optimisation et l'équilibre des ressources [1].

Google a obtenu un brevet sur la fonction MapReduce mais la validité de ce brevet est contestée, vu que « map » et « reduce » sont deux fonctions primitives de programmation qui ont été utilisées dans le développement logiciel pendant des décennies.

1.4.1.1 La fonction map

Une opération faisant appel à la fonction « map » permet de lancer une fonction pour chaque élément dans une séquence afin d'obtenir en retour une séquence de taille égale des valeurs résultantes.

1.4.1.2 La fonction reduce

Une opération faisant appel à la fonction « reduce » permet d'accumuler le contenu d'une séquence dans une seule valeur de retour, en utilisant une fonction qui va combiner chaque élément dans la séquence avec la valeur de retour de l'itération précédente.

1.4.2 Apache Hadoop

En 2009, un Framework Open Source de type Java a été créé. Ce nouveau projet Apache a été inspiré du papier publié par Google. Le traitement décentralisé des données dans Hadoop est optimisé pour des Clusters. Cette technique est déjà utilisée par des entreprises comme Yahoo, Microsoft et Facebook. Ce dernier implémente d'ailleurs en ce moment le plus grand réseau de serveurs Hadoop depuis 2010.

L'évolutivité de Hadoop permet d'améliorer les performances de traitement des données sans avoir besoin d'une connaissance de fond de son architecture. Désormais, il n'y a plus besoin de faire une montée de niveau des composants matériels. Le fait d'augmenter le nombre de serveurs de calcul dans le réseau améliore significativement le traitement des données.

1.4.3 Les bases de données non-relationnelles

Les données informatiques générées actuellement, en expansion continue, contiennent des types de données complexes et hétérogènes (texte, images, vidéos, données de géolocalisation, transactions d'achat) qui nécessitent des moteurs de traitement puissants, capables de stocker et traiter ces structures complexes de données. Les 3 V de la définition de Gartner (volume, vitesse et variété) qui décrivent cette expansion de données permettront de déduire un quatrième V, valeur du BigData. Cette valeur ajoutée met en avant le besoin de valoriser les données d'entreprises [2].

Les SGBD relationnels sont incapables de traiter cette problématique pour diverses raisons :

1- Le facteur contraignant principalement est le schéma de données, à cause des changements permanents de structure du BigData ne pouvant pas être représentés dans un schéma de données.
2- La complexité et la taille des données saturent la capacité des SGBD relationnels traditionnels pendant l'acquisition, le stockage et le traitement des données à coûts raisonnables (durée et performance de calcul).
3- La modélisation relationnelle du BigData ne s'adapte pas facilement à la persistance au morcellement des systèmes distribués.

Le NoSQL utilisé pour la première fois en 1998 est de plus en plus utilisé comme une alternative viable aux bases de données relationnelles, particulièrement pour les applications Web et les services interactifs [3] [4].

1.4.4 BigTable et HBase

BigTable est une table de données multidimensionnelle, distribuée et triée. Cette table est indexée par un identifiant de ligne, dit RowKey, un identifiant de colonne, dit ColumnKey et un horodatage, dit Timestamp, dont chaque valeur est un tableau d'octets non interprétés.

BigTable est utilisé par plusieurs applications de Google et répond à la nécessité d'un système de stockage de données structurées hautement évolutif, puisqu'il fournit un accès aléatoire aux données et en temps-réel. BigTable n'est pas une base de données relationnelle, vu qu'il structure les données dans des enregistrements agrégés dans de gigantesques fichiers indexés.

Ces enregistrements de données sont composés de colonnes groupées dans des familles. Les enregistrements de données sont identifiés par des RowKeys triés d'une manière lexicographique. Les valeurs dans chaque colonne disposent d'un Timestamp pour sauvegarder les valeurs antérieures [5]. Apache HBase créé en 2008, est l'homologue de Google BigTable au sein de Hadoop.

1.4.4.1 Apache HBase

Apache HBase est un projet lié à Hadoop. Il est actuellement utilisé dans l'application de messagerie de Facebook. Apache décrit HBase comme ayant été conçu au-dessus de de HDFS, de la même manière que BigTable est conçu au-dessus de GFS. Cette architecture est illustrée dans la Figure 4.

Comme avec beaucoup d'autres projets Apache, une communauté importante s'est créée autour de HBase [6]. La distribution HBase contient également des logiciels de cryptographie.

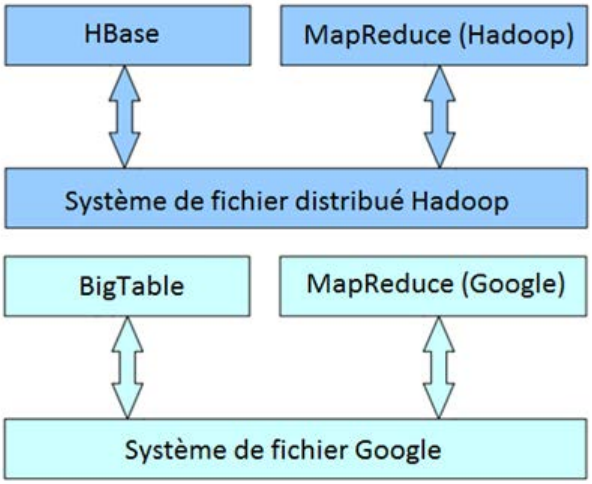

Figure 4 : Piles de Hadoop et de Google

1.4.5 GFS et HDFS

Le réseau de serveurs GFS est orchestré par un nœud maître et contient un grand nombre de serveurs de stockage. Chaque fichier est divisé en plusieurs blocs de fichiers de taille fixe de 64 MO chacun, dits Chunks. Au moment de la création, le nœud maître affecte à chacun un libellé 64-bits unique. Les Chunks sont stockés et remplacés plusieurs fois tout au long du réseau avec un minimum de trois fois. Le nœud maître stocke les métadonnées associées. Ces informations sont tenues à jour grâce aux messages de mise-à-jour de statut de chaque serveur de stockage, dits HeartBeat [7].

HDFS est l'homologue de GFS. Il est conçu pour être un système de stockage distribué, évolutif et résilient qui est conçu pour interagir facilement avec MapReduce. Il fournit une bande passante d'agrégation importante tout au long du réseau. Comme pour GFS, un réseau HDFS est composé d'un nœud maître appelé Namenode et des serveurs de données appelés Datanodes, comme expliqué dans la Figure 5. La structure du fichier HDFS est divisée en blocks de 128 MO.

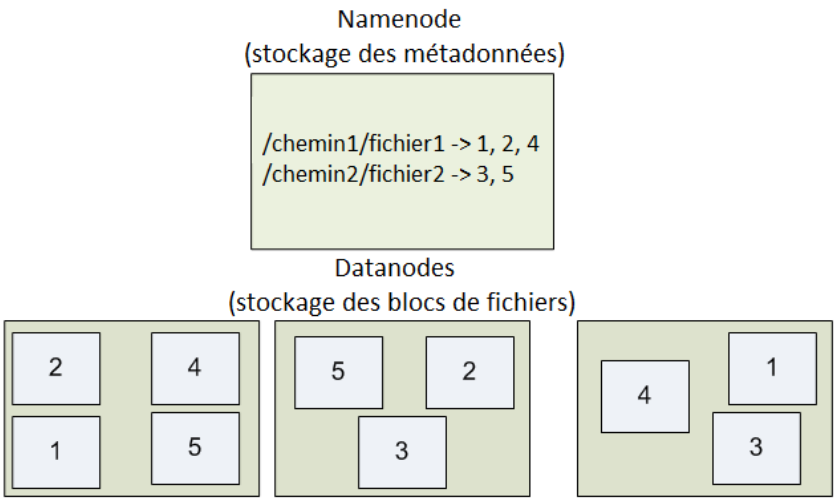

Figure 5 : Architecture HDFS

1.4.6 Conclusion

Le Framework MapReduce permet à la technologie BigData telle que fournie par Hadoop de fonctionner grâce à des composants système. Le système de fichiers HDFS utilise ces composants pour la gestion de la persistance, exécute des fonctions avancées sur les données afin de trouver des résultats en très peu de temps. Ce mécanisme consiste donc à exécuter une transformation des données, déléguée et traitée par une architecture dite Cluster et pouvant inclure des milliers d'ordinateurs qui opèrent simultanément. Hadoop possède les structures de base nécessaires pour effectuer les calculs : un système de fichiers, un langage de programmation et une méthode pour distribuer les programmes à travers le Cluster HDFS.

Avec Hadoop, le BigData est distribué en segments étalés sur une série de nœuds s'exécutant sur des périphériques de base. Au sein de cette structure, les données sont dupliquées à différents endroits afin de récupérer l'intégralité des informations en cas de panne. Les données ne sont pas organisées par rangs ni par colonnes relationnelles, comme dans le cas de la gestion classique de la persistance, ce qui comporte une capacité à stocker du contenu structuré, semi-structuré et non-structuré.

1.5 Les modèles de données non-relationnels

Les modèles de données relationnels et non-relationnels sont profondément différents. Le modèle relationnel intègre les données dans des tables interconnectées contenant des lignes et des colonnes prédéfinies. Chaque table fait référence aux autres à travers les clés étrangères stockées dans les colonnes également [8]. Lors de l'interrogation des données par l'utilisateur, l'information requise sera collectée depuis plusieurs tables et affichée. Dans l'esprit, on se pose la question de type « quel est la réponse à ma question parmi ces données ? ».

Les modèles de données non-relationnels sont souvent initiés par les requêtes applicatives, contrairement à la modélisation relationnelle. La modélisation des données non-relationnelles sera alors pilotée par des patrons d'accès applicatifs. Une compréhension avancée de la structure de données et de l'algorithme est requise afin que la logique globale soit de type « quelles questions correspondent aux données que j'ai sélectionnées ? ».

Le théorème CAP s'applique bien sur les systèmes non-relationnels. Étant donné que les modèles relationnels ont été conçus pour réagir avec l'utilisateur final, les modèles non-relationnels ont tendance à évoluer dans le but d'atteindre une structure orientée utilisateur.

On distingue cinq grandes familles parmi les types de modélisations de données non-relationnelles :

1- Le stockage clé-valeur.
2- La base de données BigTable.
3- Le modèle de données orienté document.
4- Le modèle de données orienté graphe.
5- La base de données multi-modèle.

1.5.1 Le stockage clé-valeur

Chaque enregistrement de données dans un stockage clé-valeur est présent sous la forme d'un couple composé d'une clé unique pouvant être utilisée pour identifier la valeur de données. Il s'agit du modèle non-relationnel le plus simple. Dans une base de données NoSQL grandeur nature, le stockage clé-valeur doit être partitionné à travers les systèmes distribués. Afin de s'assurer que les données sont réparties uniformément sur les partitions, plusieurs stockages clé-valeur procèdent à la technique de hachage du couple clé-valeur dans le but de déterminer l'endroit de stockage.

Un stockage clé-valeur se focalise uniquement sur la capacité de stocker et récupérer les données, plutôt que de gérer sa structure. Certains stockages clé-valeur sont optimisés pour prendre en charge les requêtes, permettant de récupérer une série d'éléments de données contigües, plutôt que des données individuelles. Ces derniers stockent régulièrement les données dans une partition selon un ordre de tri par clé, plutôt que de calculer l'endroit par la technique de hachage. Le modèle de stockage clé-valeur trié améliore significativement les performances des fonctions d'agrégation. La Figure 6 montre la disposition orientée colonne des stockages clé-valeur.

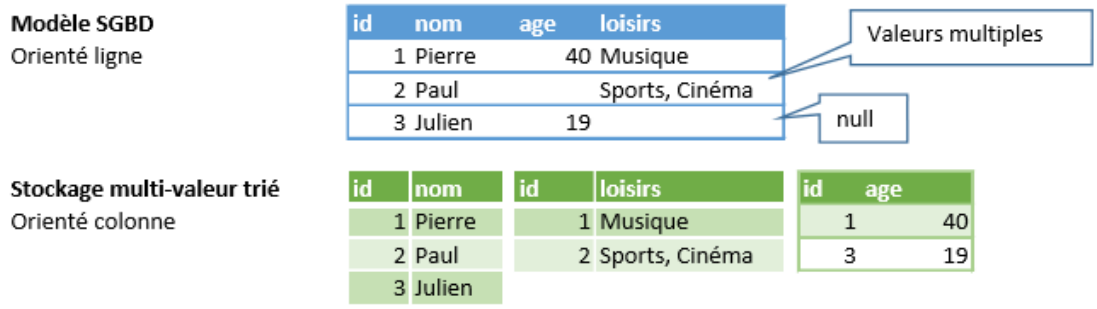

Figure 6 : Disposition orientée colonne des stockages clé-valeur

1.5.2 La base de données BigTable

La base de données BigTable contient une seule table et peut atteindre une taille immense (plusieurs PO) via un stockage distribué, à travers un grand nombre de serveurs [9]. La base de données BigTable permet de modéliser les données comme étant une carte de cartes de cartes à savoir, les familles de colonnes, les colonnes et les versions horodatées, dites Timestamps. Sa disposition orientée colonne et ses zones de stockage multi-valeur permettent de sauvegarder efficacement les données éparpillées.

Dans une base de données BigTable, les clés des colonnes ayant les mêmes caractéristiques se regroupent ensemble dans des familles de colonnes. Les colonnes d'une même famille partage en général les données de même type. Google utilise les familles de colonnes dans leur implémentation de BigTable pour stocker toutes les ancres qui pointent à la même page Web. Cette conception permet de rendre la lecture et l'écriture plus efficace dans un environnement distribué.

La composition distribuée des bases de données BigTable rend néanmoins complexe toute jointure entre deux tables. Le développeur doit implémenter cette logique dans son application ou encore la concevoir de façon à rendre cette fonctionnalité non indispensable.

Finalement dans le but de vulgariser l'usage de BigTable, Google a introduit en 2010 BigQuery, son outil permettant d'interroger les modèles de données orientés colonne. L'annexe 1 présentera une étude de la solution BigQuery de Google.

1.5.3 Le modèle de données orienté document

Une base de données orientée document est conçue pour stocker, récupérer et gérer des données orientées document ou semi-orientées document. Il s'agit d'une extension du modèle clé-valeur de façon à ce que les valeurs soient stockées dans un format structuré de document que la base de données peut interpréter. Par exemple, ce document peut être une publication sur un blog avec ses commentaires et ses Hashtags, stockés d'une façon dé-normalisée. Du moment que les données sont transparentes, ce modèle de stockage est capable de faire des opérations avancées (comme l'indexation des champs dans le document) et permet de récupérer les données d'une page Web entière avec une seule requête. Ce type est bien adapté pour les applications orientées contenu. L'avantage de la base de données orientée document par rapport au type BigTable se résume par 2 améliorations significatives (concept illustré dans la Figure 7).

1- Les valeurs avec les régimes de complexité arbitraires, pas seulement une carte de cartes.
2- L'index géré dans la base de données dans certaines implémentations.

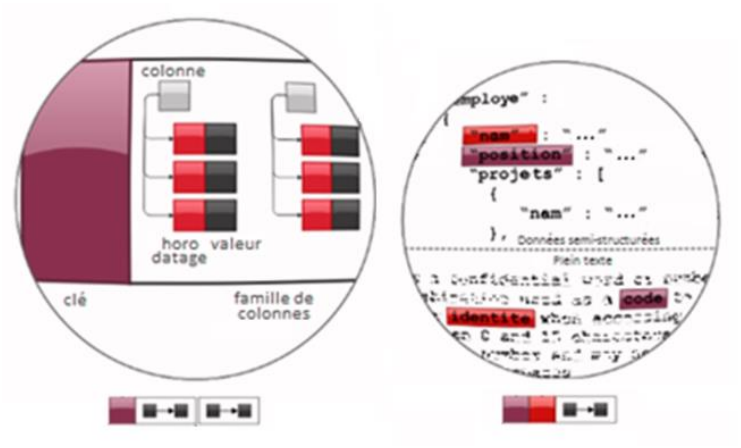

Figure 7 : Comparaison entre le modèle BigTable et le modèle orienté document

Les moteurs de recherche par mots-clés peuvent être considérés comme un type de base de données orientée document, dans le sens où ils fournissent un schéma flexible et une indexation automatique. La seule différence est que la base de données orientée document classique groupe les index par nom de champs, alors que les moteurs de recherche le font par valeur.

La modélisation orientée document suppose que l'encodage et la structuration des données se fait en formats standards, tels que le XML, YAML, JSON et BSON mais encore les types binaires, comme le PDF et l'ancien format des documents Microsoft Office.

Dans l'annexe 2, on présentera une évaluation du modèle de données orienté document de NoSQL.

1.5.4 Le modèle de données orienté graphe

Les modèles de données orientés graphe sont des bases de données non-relationnelles ne possédant pas de schéma de données. La majorité des implémentations existant de nos jours sont conformes aux propriétés ACID. Une base de données orientée graphe est composée essentiellement de nœuds, dit Nodes et d'arcs, dits Edges. Chaque nœud représente une entité (comme une personne ou une entreprise) et chaque arc représente une connexion entre deux nœuds. Chaque nœud dans ce modèle est défini par :

1- Un identifiant unique.
2- Une liste d'arcs sortant.
3- Une liste d'arcs entrant.
4- Une liste de propriétés sous forme de couples clé-valeur.

Chaque arc en revanche, est défini par :

1- Un identifiant unique.
2- Un nœud de départ.
3- Un nœud d'arrivée.
4- Un ensemble de propriétés.

Dans ce modèle de bases de données, on applique la théorique des graphes [10] aux informations de liaison entre les entités. Un exemple plus concret permettant d'illustrer explicitement ce cas de figure est celui de la liaison entre les utilisateurs dans un réseau social. Les bases de données relationnelles ne sont pas capables de stocker les informations de liaison. La récupération de ce type de données, via une requête peut être alors lente, complexe et imprévisible. Ces requêtes sont en revanche plus fiables sur une base de données orientée graphe, conçue pour un tel besoin.

Les modèles de données orientés graphe sont limités en performance dans certaines situations :

1- En parcourant tous les nœuds dans une même requête, les temps de réponse sont très lents. Les requêtes de recherche doivent être basées sur au moins une entité identifiée.
2- Dans le but d'éviter la mise à niveau du schéma de données au moment du changement du modèle, le modèle de données orienté graphe, ne possédant pas de schéma, nécessite une mise-à-jour manuelle sur tous les objets de la base.

Le modèle de données orienté graphe (exemple illustré dans la Figure 8) convient pour analyser les interconnexions, raison pour laquelle il est utilisée dans l'exploitation des données des réseaux sociaux, dite Social Data Mining. Enfin, le concept de schématiser graphiquement une base de données remonte au 18ème siècle, à travers les travaux du mathématicien Leonhard Euler.

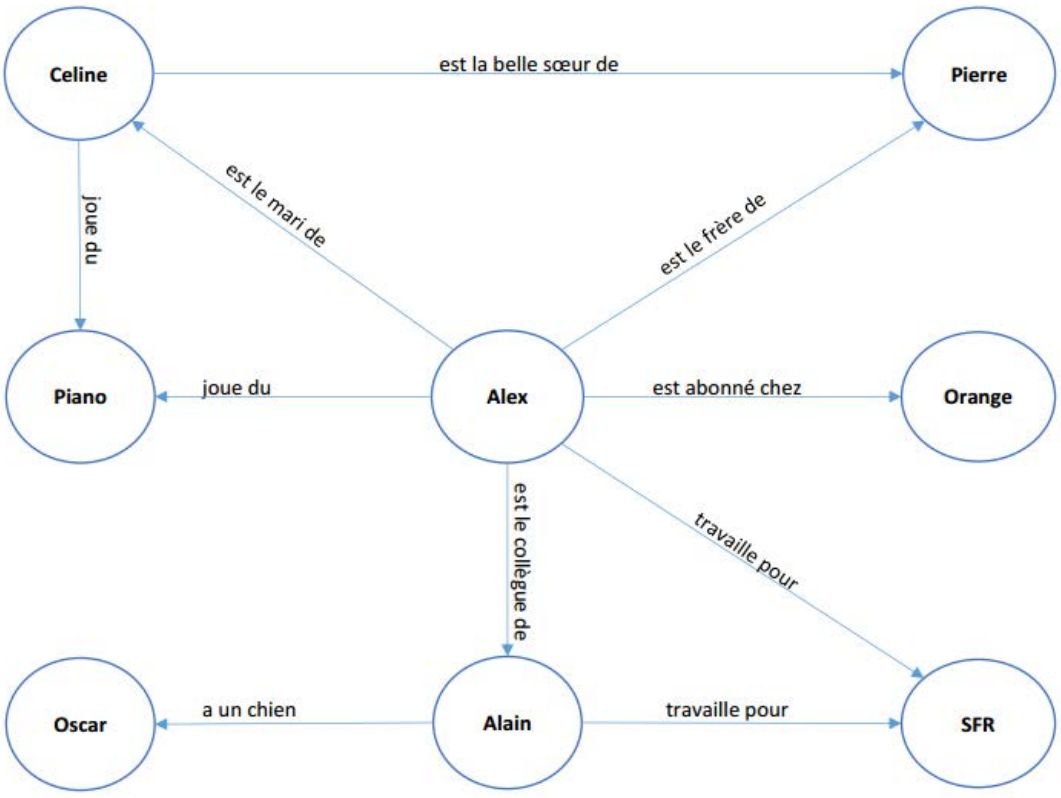

Figure 8 : Exemple du modèle orienté graphe

Dans l'annexe 3 on présentera une étude du modèle orienté graphe de NoSQL comparé au modèle relationnel.

1.5.5 La base de données multi-modèle

Avec les bases de données non-relationnelles les développeurs disposent d'un choix plus large dans la manière de structurer et de stocker les données. Il est néanmoins plus difficile de faire les bons choix car le système de stockage, le langage d'interrogation et le modèle de données sont souvent étroitement imbriqués. Les bases de données multi-modèle intègrent plusieurs modèles de données pour différentes applications au sein d'une seule et même plate-forme. Différents langages d'interrogation et modèles de données sont conservés avec le même système de stockage. Les bases de données multi-modèle sont flexibles et moins coûteuses à construire et entretenir.

Une base de données multi-modèle est la seule manière valide de satisfaire aux exigences. Il y a deux approches pour introduire cette fonctionnalité :

1- La première consiste à héberger plusieurs méthodes de stockage dans un seul système de base de données. Bien que cette technique puisse être une solution, cela revient à injecter de la flexibilité dans un système qui à la base n'est pas flexible. C'est possible mais cela nécessitera beaucoup d'efforts car il faudra éliminer ou contourner des obstacles techniques.
2- La deuxième manière consiste à élargir une base de données multi-modèle existant. Les développeurs de ces systèmes de base de données bénéficient d'un bon nombre d'avantages. Ils s'appuient toutefois sur un système de base de données qui est en principe flexible, extensible et en mesure de stocker, de parcourir et de traiter une grande variété de données.

1.5.5.1 Les avantages des bases de données multi-modèle

Les principaux avantages d'une base de données multi-modèle sont :

1- Elles offrent plus de flexibilité et présentent un meilleur rapport coût-efficacité.
2- Elles sont plus faciles à mettre à l'échelle.
3- Leur gestion est moins complexe et moins coûteuse.
4- Elles assurent la cohérence des données.
5- Elles ont un plus haut degré de tolérance aux erreurs.
6- Elles améliorent les applications dans la façon où elles présentent moins de contraintes à gérer par les développeurs.

1.5.6 Conclusion

Le NoSQL est une technologie initiée et poussée par les GAFA. Il n'y a pas de solutions parfaites et qui répondent à tous les besoins, il n'y a que des solutions adaptées aux besoins de l'utilisateur. Par ailleurs, cette approche souffre d'un manque de standardisation comme JDBC et SQL le sont pour les SGBD relationnels. Cela s'explique par la jeunesse du mouvement et l'hétérogénéité des types de bases de données existants.

1.6 L'activité principale des systèmes distribués

L'évolutivité est l'un des leviers les plus importants des bases de données non-relationnelles. Elle gère la coordination, le basculement, les ressources et d'autres capacités des systèmes distribués [11]. Les moyens engagés sont :

1- La consistance des données.
2- La création des données.
3- La coordination des systèmes.
4- La capacité à répartir la charge.
5- La tolérance aux pannes.
6- La haute disponibilité.

1.6.1 La consistance des données

La consistance des données dans les systèmes distribués est assurée à travers la réplication et la séparation spatiale des données en couple. Il en existe 3 types principaux :

1- La consistance des sauvegardes, dite Point-in-time.
2- La consistance des transactions.
3- La consistance de l'application.

En l'absence de la consistance des données, il n'y a pas de garantie qu'un quelconque morceau de donnée dans le système est uniforme à travers le réseau. Les questions de consistance se posent même au sein d'un environnement comportant une seule machine, notamment pendant la restauration des données de sauvegarde et leur utilisation à la place des données originales. Le principal avantage de la consistance des données est de maintenir l'intégralité des informations stockées. Maintenir la consistance des données est un objectif essentiel des logiciels de traitement des SGBD.

1.6.2 La création des données

Les algorithmes de création des données gèrent la liaison entre les éléments de données et les nœuds physiques, la migration des données d'un nœud à un autre et l'allocation globale des ressources dans la base. Le mécanisme principal pour la création des données consiste à fournir un modèle d'abstraction indépendant et des règles de migration pour le déplacement et le calcul des données entre les serveurs. Ce mécanisme ne gère pas en revanche les décisions qui relèvent de contraintes juridiques ou opérationnelles nécessitant la création de duplicatas des données [12].

1.6.3 La coordination des systèmes

Les bases de données distribuées nécessitent la coordination des activités entre le nœud principal et les autres serveurs à travers le réseau. Si le nœud principal se met en échec, alors un autre serveur doit prendre sa place (selon un algorithme de sélection bien défini). Apache ZooKeeper fournit un ensemble de règles d'usage fiables et de blocs de construction permettant d'assurer la coordination des systèmes.

1.6.4 La capacité à répartir la charge

Afin d'augmenter la capacité de traitement d'un système distribué, il est souvent moins coûteux de multiplier le nombre de serveurs physiques plutôt que d'augmenter la capacité de traitement de ceux-ci. Cette démarche entre dans le cadre de l'évolutivité du Cluster. En effet, l'évolution du coût d'un serveur informatique en fonction de l'évolution de sa capacité de traitement répond à une fonction de type exponentielle. Plusieurs solutions agissant au niveau physique ou logique permettent de gérer la répartition de charge entre différents serveurs.

1.6.5 La tolérance aux pannes

Les systèmes distribués étant constitués d'un ensemble de composants répartis sur plusieurs machines physiques, ces composants comportent une couche de communication permettant d'assurer la liaison entre les autres composants. Cette prédisposition à la communication inter-composant permet aux systèmes distribués d'implémenter plus facilement des solutions de haute disponibilité par exemple en redondant un composant logiciel sur plusieurs serveurs physiques afin de permettre à un deuxième serveur de reprendre la main sur le premier en cas de défaillance de celui-ci.

La majorité de ces solutions imposent tout de même une certaine latence dans la reprise en charge des requêtes d'un serveur défectueux dite Failover par un serveur en bon état. De plus, les requêtes envoyées au serveur au moment du plantage ou pendant ce temps de latence ne seront pas traitées par le système. On constate tout de même que quelques solutions adaptées au transport des messages proposent des mécanismes de tolérance aux pannes garantissant l'absence de perte de requête tout en réduisant fortement le temps de latence lors du basculement.

1.6.6 La haute disponibilité

Pour certains composants d'un système distribué la charge subie consiste uniquement en des pics occasionnels. Le composant connaîtra un ou plusieurs pics d'activité dans une journée : c'est typiquement le cas pour un composant prenant en charge l'authentification des utilisateurs et le rapatriement des informations de session, ce composant sera sollicité très fortement en début de journée pour l'ouverture des sessions et en fin de journée pour la fermeture.

Pour répondre à ces phases de sollicitation importantes on redonde le composant en question sur plusieurs machines physiques grâce aux mécanismes de répartition de charge vu précédemment. Seulement, il s'avère qu'avec une répartition de charge classique, le partage des ressources entre les différents composants reste statique. On définit pour chaque composant le nombre d'instances que l'on souhaite créer en fonction de la capacité de traitement cible de ce composant. Or dans notre cas, où le composant est sollicité uniquement par pic d'activité, ce composant va consommer des ressources inutilement lorsqu'il ne sera que très faiblement sollicité.

Pour permettre de répondre à cette problématique, de nouvelles solutions, permettant l'instanciation et la destruction en temps-réel d'un composant logiciel, ont vu le jour.

1.6.7 Les difficultés de mise en œuvre

La mise en place de systèmes distribués engendre un certain nombre de difficultés lors de la mise en œuvre, dont :

1- La gestion de l'hétérogénéité en termes de synchronisation des horloges et de cohérence des données.
2- La gestion des composants et le fonctionnement en mode dégradé.
3- La disponibilité et la détection des arrêts.
4- La gestion de la séquentialité.

1.6.7.1 La gestion de l'hétérogénéité

Lors de la mise en place d'un système distribué il est nécessaire que l'ensemble des composants travaille avec des données cohérentes. Cette cohérence des données est d'autant plus problématique lorsqu'on redonde certains composants pour augmenter la capacité de traitement ou la disponibilité du système. En effet, les données, comme le cache applicatif, le contenu d'une base de données ou bien les variables de session des utilisateurs Web doivent être synchronisées entre les différentes instances d'un composant afin d'assurer une cohérence dans les traitements réalisés.

1.6.7.2 La gestion des composants

Un système distribué regroupant un ensemble de composants logiciels répartis sur plusieurs serveurs physiques, il est nécessaire pour assurer la maintenance corrective et évolutive du système, de dresser sa cartographie complète. Il est difficile de prévoir les impacts au niveau des différents systèmes à la suite d'un changement sur l'un des composants.

1.6.7.3 La disponibilité et la détection des arrêts

Dans un système distribué, l'indisponibilité d'un seul composant du système peut rendre indisponible le système complet. On mesure alors la disponibilité de ce type de système à celle de son maillon le plus faible.

Pour couvrir ce risque, il est nécessaire de mettre en place en amont une architecture physique permettant d'assurer la disponibilité cible pour tous les composants. Une fois que cette architecture est en production, des opérateurs doivent à l'aide de logiciels spécifiques, s'assurer de la détection au plus tôt d'une défaillance de l'un des composants de l'architecture.

1.6.7.4 La gestion de la séquentialité

La mise en place d'un Cluster provoque la création de deux points d'entrée au système. Dans le cas d'un système distribué d'échange de données par exemple, Il est possible que deux modifications successives du même objet soient dirigées vers deux nœuds différents du Cluster, ce qui dans l'absolu peut aboutir à une situation où le message le plus récent est diffusé en premier vers l'application destinatrice. Si aucune gestion de la séquentialité des messages n'est gérée, le message le plus ancien viendra écraser dans les applications destinatrices, le message le plus récent.

1.6.8 Conclusion

Les systèmes distribués offrent un certain nombre de moyens permettant d'augmenter la performance globale du système et son niveau de disponibilité. Cependant, le choix du niveau de distribution, ainsi que du niveau de redondance des composants permettant de déterminer la performance et le niveau de disponibilité global du système, reste à la charge de l'utilisateur qui met en place ce type de système. Ce choix doit être réalisé en comparant les contraintes d'exploitation métier et le budget alloué à la mise en place du système. La recherche de la performance sans réelle contrainte métier n'a pas de sens.

Un système distribué permet alors d'atteindre des niveaux de performance et de disponibilité importants plus facilement et à moindre coût, grâce à l'utilisation de plusieurs serveurs. Ceci est à comparer avec les systèmes centralisés, se basant sur l'utilisation d'un gros serveur unique ayant une capacité de traitement supérieure.

1.7 Conclusion du chapitre

Les volumes BigData engendrent une évolution considérable des modèles technologiques, une évolution qui permet d'accéder à de nouvelles opportunités et de mieux contrôler les risques. Le BigData représente certes une opportunité significative mais pose aussi des défis spécifiques. Ces derniers incluent un ensemble relativement nouveau de technologies plutôt complexes à appréhender, dépourvues pour l'instant d'outils pour encourager leur adoption et leur développement.

Par ailleurs, les ressources documentées sont peu nombreuses. L'approche Open Source de certaines solutions, ainsi que les plate-formes flexibles d'intégration pour le BigData relèvent ces défis, permettant aux utilisateurs de relier facilement et analyser des données provenant de systèmes disparates dans le but de contribuer à piloter et améliorer la performance de leurs applications.

Chapitre 2. La problématique étudiée

2.1 Introduction au chapitre

BigData est devenu un fait dans le monde d'aujourd'hui, une vraie thématique à gérer en termes d'infrastructure et une nécessité dans notre quotidien. Cette quantité énorme de données pourrait se prêter à des traitements par des algorithmes de traitement efficaces :

1- Comme une base de départ pour la création de nouvelles entreprises.
2- Pour analyser efficacement le comportement des utilisateurs, savoir ce qu'ils cherchent, combien de temps à passer sur Internet.

Ces données sont en incrémentation tous les jours grâce à l'évolution de la science et le progrès technique dans le domaine de l'informatique. Néanmoins cette quantité brutale de données requiert de meilleurs algorithmes de traitement et des systèmes plus performants permettant de résoudre les problèmes plus facilement. De plus ces quantités de données dépassent les capacités des machines individuelles et nécessitent des milliers de machines travaillant ensemble et assurant un traitement organisé avec MapReduce.

Dans le chapitre précédent, on a décrit le modèle de stockage non-relationnel en se penchant sur les moteurs de traitement existant aujourd'hui et leur efficacité. Ensuite on a décrit les différents types de bases de données non-relationnelles, ainsi que l'activité principale des systèmes distribués.

Le présent chapitre décrit le modèle de programmation MapReduce en termes d'historique, de prérequis, d'implémentations, d'algorithmes utilisés, d'applications dans notre quotidien et de gains réalisés. Les pages suivantes, aborderont les concepts de base de plusieurs types de traitement (graphes, indexation, données textuelles) tout en illustrant les propos théoriques par des exemples de la vie quotidienne. Les derniers paragraphes décriront une explication détaillée des algorithmes EM et de leur usage. Finalement, on évoquera les principaux projets appartenant à l'univers de Hadoop MapReduce tels que YARN, Apache Storm et Apache Spark.

2.1.1 MapReduce et Cloud Computing

Toute application qui s'exécute dans un navigateur Web et qui génère des données utilisateur (messagerie, réseaux sociaux, partage de médias) dispose d'un périmètre de traitement dans le Cloud. Typiquement il s'agit d'un traitement ciblant les comportements utilisateur envers une publicité ou encore les propositions de connexions d'amis dans les réseaux sociaux et qui génèrent des données importantes sur les serveurs et nécessitent des algorithmes de traitement optimisés comme MapReduce.

2.1.2 Les idées de départ

MapReduce est le fruit d'un rassemblement d'idées qui ont longtemps existé séparément et qui mises ensemble, ont prouvé leur efficacité.

2.1.2.1 L'évolutivité des applications

En cas de ressources insuffisantes sur un serveur donné, il est conseillé de procéder par une évolution par ajout d'un autre serveur (Scale Out), plutôt qu'une évolution par augmentation des capacités du matériel sous-jacent (Scale Up) dans le but d'obtenir une basse consommation en énergie (plusieurs serveurs chargés chacun à 10% consomment moins d'énergie et nécessitent statistiquement moins d'entretien qu'un seul serveur chargé à 100% en permanence). Cela signifie bien entendu une architecture spécifique des applications permettant d'utiliser efficacement les ressources matérielles disponibles.

2.1.2.2 Les défaillances fréquentes et inévitables

Un matériel électronique peut tomber en panne à tout moment. Cela est valable pour une simple barrette de mémoire ou d'un disque dur pouvant bloquer un serveur mais aussi pour un routeur capable de paralyser toute une partie du Cluster. Un algorithme bien conçu doit faire face aux défaillances en redistribuant automatiquement la charge de travail sur les ressources encore disponibles sans affecter la qualité du service fourni.

2.1.2.3 Le traitement décentralisé

Afin d'éviter les transferts de données inutiles sur le Cluster, il est conseillé de procéder à un traitement ciblé dans le processeur lié au bloc de données correspondant. En d'autres termes, si les données se trouvent dans des nœuds séparés, il suffit de déplacer le programme de traitement et de le lancer sur chaque nœud, plutôt que de déplacer toutes les données sur un seul serveur de traitement. Ce fonctionnement est décrit dans l'exemple de la Figure 9.

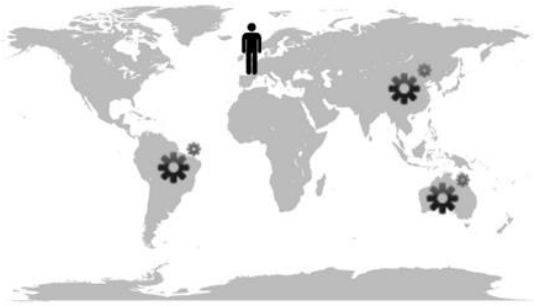

Figure 9 : Exemple de traitement MapReduce

2.1.2.4 Le traitement séquentiel non aléatoire

Pour mettre à jour 1% des enregistrements d'un bloc de données de taille importante, il prendrait beaucoup moins de temps en général de le lire entièrement puis de le réécrire en modifiant les enregistrements de données en question, plutôt que d'effectuer des modifications unitaires à chaque accès aléatoire aux données. Cela nécessite en revanche un matériel adapté et très onéreux.

Le traitement MapReduce consiste à séquencer les données à traiter, puis de rassembler les valeurs par la suite pour fournir un résultat final. Par exemple, dans la Figure 10 on décrit le fonctionnement d'un algorithme MapReduce pour le calcul du nombre de mots dans un ensemble de fichiers. Le traitement séquentiel s'exprimera alors comme suit :

1- Pour chaque fichier d'une séquence de fichiers, lire le contenu du fichier dans une chaîne de caractères, compter les mots dans la chaîne de caractères et ajouter ce compte au cumul du nombre total de mots dans la séquence.
2- Retourner le cumul total une fois la séquence source épuisée.

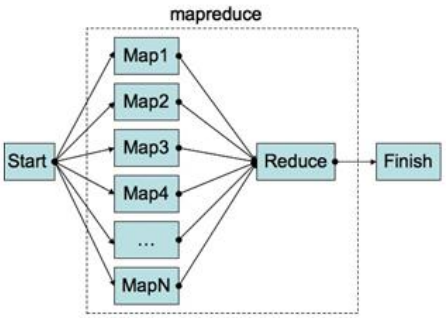

Figure 10 : Traitement séquentiel

2.1.2.5 Le développeur isolé de l'environnement

Dans le but de faciliter l'écriture du code, le développeur n'a plus à se préoccuper de la performance du système qui est du ressort du Framework d'exécution, validé d'avance. Cela lui permettra de se concentrer plutôt sur l'algorithme de fonctionnement de l'application. La Figure 11 représente l'utilisation du Framework MapReduce, comme une boîte noire avec des données en entrée et sortie.

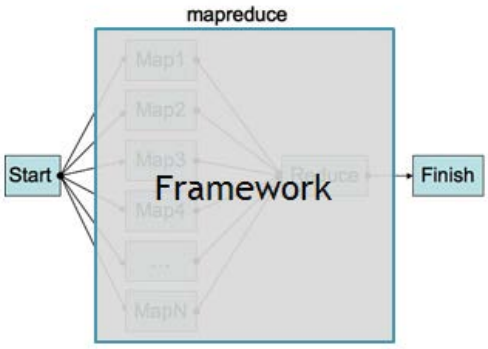

Figure 11 : Framework MapReduce

2.1.2.6 L'évolutivité en souplesse

Un algorithme de traitement de complexité O(n) prendrait deux fois plus de temps à traiter des quantités de données deux fois plus importantes. En revanche, ce même algorithme pourrait idéalement prendre la moitié du temps nécessaire, si on doublait la taille du Cluster [13]. Cet algorithme devrait s'exécuter idéalement sans nécessiter une modification ou reconfiguration, tout en s'adaptant au changement de circonstances. Cet exemple est illustré dans la Figure 12.

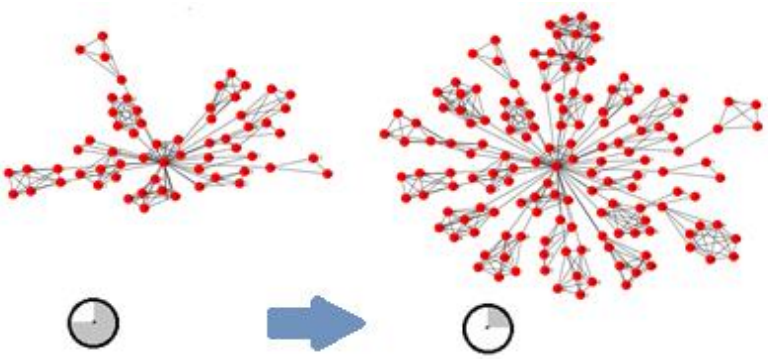

Figure 12 : Evolutivité en souplesse

2.1.3 L'importance du MapReduce

Ce modèle de programmation dont le mécanisme est représenté dans la Figure 13 est un outil efficace pour résoudre les problèmes de données massives mais a également changé l'organisation des systèmes de traitement en large échelle. MapReduce est une liaison entre les deux couches physique et logicielle. Malgré le succès qu'il a eu auprès de millions d'utilisateurs dans le monde d'aujourd'hui [14], ce modèle est limité et loin d'être dans son aspect final. Au contraire, il s'agit d'un point de départ vers d'autres perspectives.

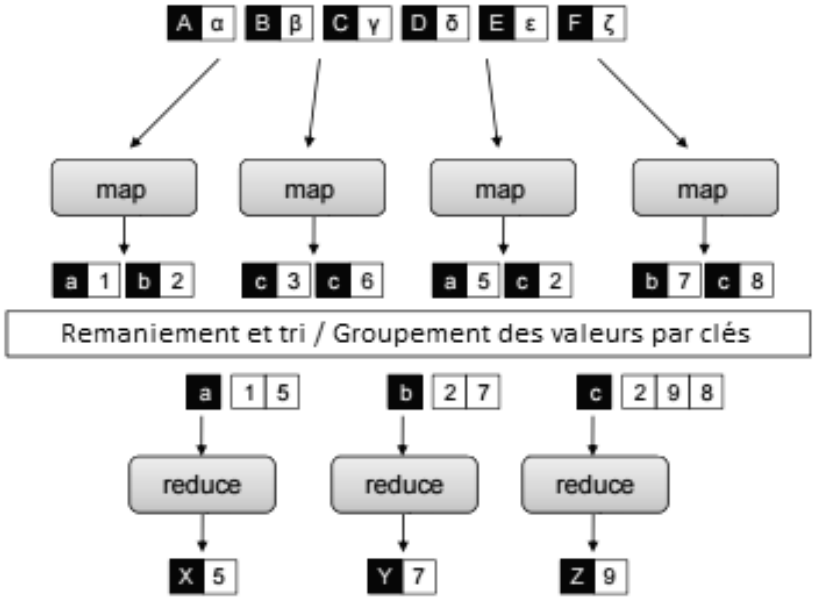

Figure 13 : Vue simplifiée de MapReduce

2.2 Les notions de base

MapReduce applique le principe de base « Diviser pour Régner ». En d'autres termes, diviser un gros problème en plusieurs petits problèmes et les répartir sur les multitudes de processeurs disponibles dans le Cluster pour traitement. MapReduce dispose d'un avantage significatif par rapport à d'autres algorithmes de parallélisme comme OpenMP et MPI grâce à son Framework d'exécution qui possède la capacité de s'occuper de toute la partie environnement système et communications bas niveau.

Plusieurs implémentations de MapReduce coexistent et les plus importantes sont celles de Google et de Hadoop. Ce dernier étant la solution Open Source choisie par la majorité des développeurs. Des différences existent néanmoins entre les deux implémentations, notamment au niveau du tri des valeurs, encore indisponible dans certaines version de Hadoop ou de l'attribution sélective des clés par le développeur, non autorisée dans l'implémentation de Google.

Les « Mappers » et « Reducers » sont les deux types d'objets implémentant les méthodes « map » pour le morcellement et « reduce » pour le regroupement, respectivement et dans chaque itération dans le modèle MapReduce. La présence des « Mappers » est obligatoire dans un programme implémentant ces modèles, contrairement aux « Reducers ». Dans la solution de Google, BigTable contient les données d'entrée du programme [5] et sert à stocker les données de sortie. Son équivalent dans l'univers Hadoop est la solution Open Source HBase, possédant des capacités similaires.

2.2.1 Le Framework d'exécution

Une fois que le développeur a soumis un programme au nœud de départ dans le Cluster, les composants du Framework prennent en charge tous les aspects d'exécution jusqu'à la sortie.

2.2.1.1 La planification

Dans le cas où le nombre total des tâches à exécuter dépasse le nombre de nœuds disponibles dans le Cluster d'exécution, le Framework MapReduce planifie l'exécution des tâches en attente et les lance au fur et à mesure de la disponibilité des nœuds. Cette planification a lieu également au moment de l'exécution de tâches appartenant à des programmes différents.

2.2.1.2 La relocalisation du code

Il s'agit de déplacer le code d'exécution vers les nœuds plutôt que de déplacer les données. Pour un meilleur traitement, le code est déplacé vers le nœud contenant le bloc de données à traiter. En cas de surcharge sur un nœud donné, une partie des données sera déplacée vers un nœud disponible. Les nœuds les plus proches sont naturellement prioritaires.

2.2.1.3 La synchronisation

Toute tâche « map » est suivie éventuellement par une tâche « reduce ». La synchronisation correspond à la copie des données de la sortie « map » vers l'entrée « reduce ». Cette opération est essentielle et a lieu à la fin de chaque tâche « map ».

2.2.1.4 La gestion d'erreur

Dans un processus MapReduce à grande échelle certaines erreurs sont inévitables (problèmes de mémoire [15] ou d'écriture sur le disque [16]). Une gestion propre des faux enregistrements dans les données traitées est nécessaire.

2.2.2 L'architecture de la couche de données

Le nombre de nœuds de calcul est proportionnel à la capacité des volumes de stockage utilisés. Par conséquent, de bonnes performances de traitement nécessitent un investissement important. Les systèmes de fichiers GFS (Google) ou HDFS (Hadoop) utilisent l'architecture classique client-serveur dans laquelle le nœud maître appelé Namenode, contenant la structure des répertoires et les permissions, gère les accès vers les nœuds esclaves appelés Datanodes, où sont stockés les blocs de données. Trois copies différentes des blocs de données sont créées à chaque fois dans des endroits différents du Cluster afin de réagir en cas de défaillance matérielle.

Le Namenode est en contact permanent avec les Datanodes à travers des messages spécifiques afin de maintenir le système de fichiers en bonne santé. Une redistribution de charge de travail est appliquée afin d'éviter tout blocage dû à une surcharge dans certains Datanodes. Il est conseillé de travailler avec des fichiers de grande taille afin de minimiser la quantité d'information à gérer dans le Namenode (stockée en mémoire vive pour des accès plus performants). Cette architecture simplifiée, dite Single-Master, possède une faille malgré tout dans le cas où le Namenode tombe en panne : tout le système de fichiers s'écroule. Afin de contourner ce problème, il est possible de configurer plusieurs Namenodes dans le système, prêts à prendre le relais en cas de défaillance. Cette architecture est décrite dans la Figure 14.

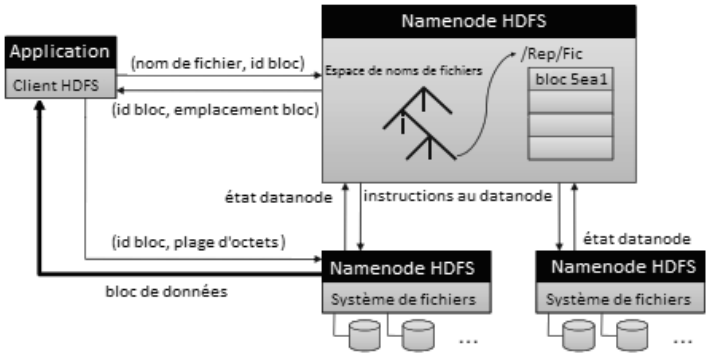

Figure 14 : Architecture HDFS

2.2.3 Conclusion

Le modèle de programmation MapReduce fournit un cadre au développeur lui permettant d'écrire des fonctions « map » et « reduce ». Tout l'intérêt de ce modèle est de simplifier la vie du développeur. Ainsi, il n'a pas à se soucier du travail de parallélisme et de distribution, de récupération des données sur HDFS, de développements spécifiques à la couche réseau pour la communication entre les nœuds ou d'adapter son développement en fonction de l'évolution de la montée en charge à travers le Cluster. Ainsi, le modèle de programmation MapReduce permet au développeur de ne s'intéresser qu'à la partie algorithmique. Il transmet alors son programme développé dans un langage de programmation au Framework MapReduce pour l'exécution.

2.3 Le concept MapReduce

L'implémentation de l'algorithme MapReduce par le développeur se limite à la création du « Mapper » et du « Reducer ». Tous les autres aspects sont gérés par le Framework d'exécution. Cependant le développeur dispose de plusieurs techniques lui permettant de contrôler et de gérer les flux de données :

1- Construire des données structurées en entrée / sortie.
2- Exécuter des commandes spécifiques en entrée / sortie.
3- Préserver l'état des « Mappers » et des « Reducers » tout au long des nœuds.
4- Contrôler l'ordre de traitement des valeurs.
5- Contrôler la répartition des clés entre les « Mappers » et les « Reducers ».

2.3.1 Les patrons de conception

Lors du traitement des données massives, l'aspect de synchronisation le plus important est l'échange des résultats intermédiaires, c'est-à-dire le transit des données sur le réseau. Ce traitement fait l'objet de l'utilisation de plusieurs techniques illustrées dans la Figure 15.

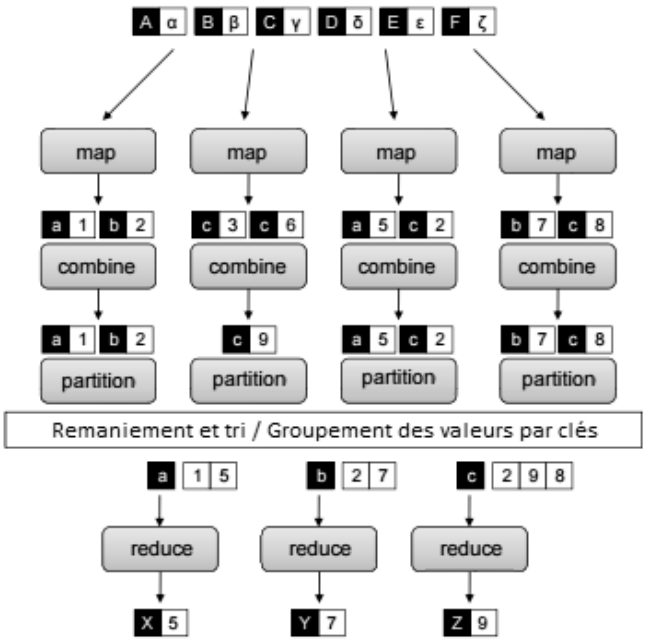

Figure 15 : Vue complète de MapReduce

2.3.1.1 La technique In-Mapper Combining

Des « Combiners » jouant le rôle de « Reducer » à une petite échelle et permettant de faire des agrégations afin de synthétiser les données échangées, sont souvent utilisés. Il est possible d'optimiser davantage ce fonctionnement avec la technique In-Mapper Combining qui permet de réduire d'une manière significative les combinaisons clé-valeur transmises entre « Mappers » et « Reducers », contrairement aux « Combiners ». Cependant, l'utilisation de cette technique a tendance à provoquer des problèmes de saturation de mémoire qui peuvent néanmoins être résolus en faisant des purges de mémoire régulièrement.

2.3.1.2 Les techniques Pairs et Stripes

D'autres techniques telles que Pairs et Stripes permettent de modéliser l'apparition d'événements joints et offrent une alternative entre passage à l'échelle et performances, sachant que Stripes est la plus performante et la plus rapide.

Finalement, le modèle « Order Inversion » utilisé dans de nombreuses applications, permet aux fonctions « reduce » d'accéder à des valeurs intermédiaires avant de traiter les données qui ont généré ces valeurs.

2.3.1.3 Le tri secondaire

MapReduce trie les couples clé-valeur par ordre alphabétique de clé, ce qui permet d'optimiser les performances de traitement et surtout si les calculs du « Reducer » s'appuient sur un ordre de tri. L'implémentation de Google offre la possibilité d'appliquer un tri secondaire optionnel au niveau des valeurs. Cette fonctionnalité inexistante dans Hadoop permet d'optimiser davantage le traitement.

2.3.1.4 La conversion des valeurs en clés

Cette technique, dite Value-to-Key Conversion est un patron de conception dont l'idée de base consiste à déplacer une partie de la valeur dans une clé intermédiaire pour former une clé composite. Le tri sera géré par la suite par le Framework d'exécution. Cette technique non seulement implémente facilement le tri secondaire mais peut être étendue vers d'autres tris.

2.3.2 Les jointures relationnelles

L'entrepôt de données dit « Data Warehouse » est un outil de Business Intelligence. Il s'agit de l'une des applications les plus connues de Hadoop. Il contient des données de nature relationnelle mais sert aussi à stocker des données semi-structurées ou non-structurées. Les cubes OLAP en sont un exemple d'implémentation.

Plusieurs solutions basées sur Hadoop et devenues Open Source par la suite existent, permettant de faire des traitements de Business Intelligence, telles que Hive et Pig [17]. MapReduce dispose donc par ce biais des algorithmes nécessaires pour traiter des données relationnelles.

2.3.2.1 La jointure Reduce

Cette jointure, dite Reduce-Side Join, nécessitant un tri secondaire, est très coûteuse au niveau de la mémoire. La création d'une clé composite et l'application du patron de conception Value-to-Key Conversion peut être une solution mais cela signifie que les Datasets de données doivent être mixés partout sur le réseau, ce qui n'est pas très performant.

2.3.2.2 La jointure Map

Contrairement à la jointure Reduce, il existe aussi la jointure Map, dite Map-Side Join. Il s'agit d'une jointure qui est beaucoup plus performante, vu que les Datasets où s'applique la jointure, se retrouvent à la sortie d'un traitement précédent donc aucune ressource supplémentaire à allouer pour faire ce traitement.

2.3.2.3 La jointure en mémoire

Cette dernière technique appelée également Memory-Backed Join peut être appliquée quand un des deux Datasets est capable de tenir en mémoire de chaque nœud de traitement. Dans ce cas, le plus petit Dataset est utilisé pour faciliter les accès aléatoires basés sur la clé de la jointure. Dans le cas où les deux Datasets sont trop gros pour tenir en mémoire, il est possible de partitionner le plus petit des deux en plusieurs morceaux. Une autre approche consiste à stocker ce Dataset dans la mémoire collective de plusieurs nœuds en parallèle [18].

2.3.3 Conclusion

Le Framework MapReduce fournit au-delà des composants « Mapper » et « Reducer », une liste de techniques permettant d'implémenter des méthodes de programmation plus avancées, telles que les jointures, la fusion, le tri ou d'autres.

2.4 Le traitement par indexation inversée

En tapant un mot clé dans un moteur de recherche sur Internet, l'utilisateur s'attend à un temps de réponse relativement court, voire instantané. Ces exigences doivent être respectées malgré la quantité gigantesque de données textuelles qui se trouve sur le Web. Pratiquement tous les moteurs de recherche aujourd'hui utilisent la technique d'index inversé, fournissant un accès à la liste des documents pouvant contenir le mot clé tapé par l'utilisateur.

La recherche sur le Web nécessite donc 3 étapes afin de pouvoir afficher des résultats :

1- Le rassemblement du contenu Web en indexant les pages, dit Crawling.
2- La construction d'index inversés, dite Inverted Index.
3- Le classement des documents dans une requête, dit Retrieval.

Les deux premières étapes partageant beaucoup de caractéristiques, doivent être évolutives et efficaces mais ne nécessitent pas de traitement en temps-réel. Elles tournent régulièrement en fonction de la fréquence de mise-à-jour des pages. En ce qui concerne la troisième étape, le Retrieval, c'est tout à fait le contraire, vu qu'une réponse en temps-réel est attendue.

2.4.1 L'indexation

Ce processus consiste à parcourir le Web à travers les liens hypertexte et à stocker les pages téléchargées pour un traitement ultérieur. L'idée en soi ne paraît pas compliquée; cependant, certaines contraintes doivent être respectées :

1- Ne pas surcharger le serveur Web pendant le traitement.
2- Établir l'ordre de téléchargement des pages non visitées.
3- Ne pas télécharger plusieurs fois la même page sur des nœuds séparés.
4- Pouvoir surmonter les problèmes de réseaux et les défaillances matérielles.
5- Choisir la bonne fréquence de téléchargement de la page afin de rafraîchir son contenu.
6- Identifier les duplicatas de documents et télécharger le contenu le plus fiable pour l'indexer.
7- Prendre en considération le caractère multilingue lors du passage d'une page à l'autre.

2.4.2 L'indexation inversée

Il s'agit d'un ensemble de listes de correspondance dont chacune correspond à un mot clé. Chaque liste contient plusieurs couples clé-valeur dont la clé est l'identifiant du document et la valeur est la fréquence d'apparition du mot clé dans ce document, comme indiqué dans la Figure 16. Certains moteurs stockent également dans la valeur d'autres informations, telles que la position de chaque occurrence ou le format utilisé (exemple des liens hypertexte). De nombreux travaux de recherche dans ce domaine visent à optimiser ce fonctionnement [19].

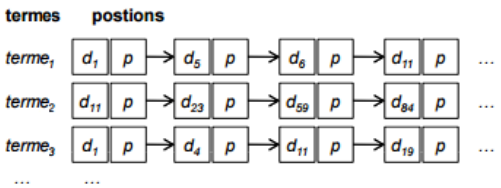

Figure 16 : Illustration simple des index inversés

2.4.2.1 La mise en œuvre

Le modèle de programmation MapReduce fournit une implémentation facile de cette technique. Dans ce modèle les documents subissent un traitement parallèle. Les « Mappers » se lancent avec comme variable d'entrée les clé-valeurs identifiants des documents couplés aux contenus Web. Le traitement varie en fonction de l'application ou du type de document :

1- Les balises HTML, ainsi que le code JavaScript ne sont pas pris en compte.
2- Les mots communs et les affixes sont ignorés également.

Les éléments à la sortie sont triés par identifiant de document, pour constituer une nouvelle liste de correspondance. Au final, ces listes d'index inversés seront écrites sur le disque.

2.4.2.2 L'optimisation

Le Framework d'exécution ne garantit pas le tri des valeurs associées à la même clé. Le « reduce » se met à faire une lecture des données, puis un tri en mémoire avant de procéder à l'écriture de la sortie sur le disque. Ce qui pourrait poser des problèmes de saturation de mémoire pour les très longues listes. Il existe une solution simple pour ce problème qui consiste à déléguer le tri au Framework d'exécution, grâce à la technique de conversion des valeurs en clé, dite Value-to-key Conversion.

2.4.2.3 La compression des index et des valeurs

Toute compression permet de gagner de l'espace disque, par conséquent, du temps au moment de l'écriture sur le disque. En revanche, elle nécessite des ressources supplémentaires pour coder et décoder les données. Autrement dit, un temps de traitement supplémentaire. Un certain équilibre entre les deux permet d'obtenir un meilleur résultat. Des algorithmes de compression tels que Bit-Aligned et Word-Aligned peuvent être utilisés.

Les valeurs peuvent être compressées de la même manière en prenant en considération les caractéristiques de la collection pouvant affecter les performances de traitement. La Figure 17 illustre l'utilisation des index inversés selon un algorithme MapReduce basé sur ces techniques.

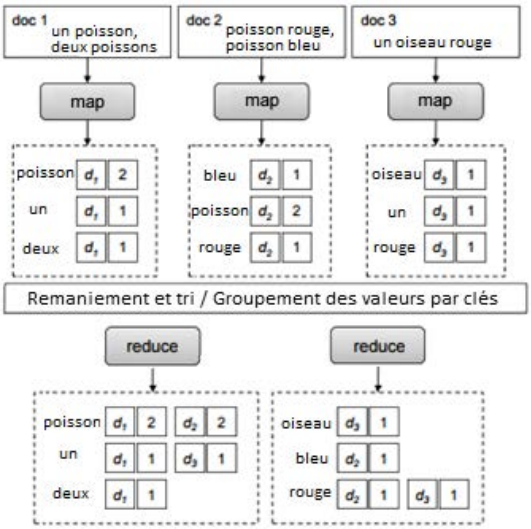

Figure 17 : Algorithme avancé des index inversés

2.4.3 Le classement

Quel que soit le nombre d'utilisateurs qui lance des requêtes sur un moteur de recherche, ils s'attendent néanmoins à des résultats instantanés. Cela dépasse largement les capacités d'une seule machine. La seule solution est de distribuer la charge de travail sur un grand nombre de serveurs. Il existe principalement 2 stratégies de partitionnement pour une récupération distribuée :

1- Par document, en découpant la collection en plusieurs sous-collections, chacune prise en charge par un serveur.
2- Par mot clé, où chaque serveur traite une partie des mots clés en question.

Les études ont démontré que le partitionnement par document est la meilleure stratégie [19]. C'est d'ailleurs celle adoptée par Google.

2.4.4 Conclusion

L'indexation inversée est la technique la plus fréquente d'indexation dans MapReduce. Elle agit de façon à ce que chaque terme de l'index soit décrit par le numéro de référence de tous les documents qui contiennent ce terme et la position du terme dans ce document. Elle permet une accélération considérable de la recherche pour une requête.

L'index créé peut être ordonné en fonction décroissante de la fréquence des termes. Il est construit séquentiellement en lisant les documents. Vu qu'il est trié par termes et documents, différentes informations peuvent être alors associées aux mots du dictionnaire (la fréquence documentaire par exemple). Les documents par la suite sont arrangés en une liste triée par document ou par score (étant dépendant du modèle).

Cette technique s'implémente dans différentes structures de données, notamment les tris (stockage des chaînes de caractère dans des arbres) ou les tables de hachage.

2.5 Le traitement des graphes

Les graphes sont omniprésents dans notre entourage :

1- Les liens hypertexte dans un site Web.
2- Les réseaux sociaux (connexions d'amis, historiques d'appels ou de mails).
3- Les réseaux de transports (routes, lignes, voyages).

En général les graphes sont composés de nœuds et d'arcs directs ou indirects connectant ces nœuds. Il se peut qu'un nœud soit lié à lui-même. Les nœuds et les arcs peuvent êtres tous les deux annotés par des métadonnées.

2.5.1 L'application

Des algorithmes de traitement peuvent être appliqués aux graphes pour résoudre beaucoup de problématiques dans le monde d'aujourd'hui :

1- En recherche et planification, on dispose d'algorithmes fréquemment utilisés dans notre quotidien, comme pour la recherche de trajets dans les réseaux de transports ou la suggestion d'amis dans les réseaux sociaux.
2- En classification, il est possible de diviser un graphe en plusieurs morceaux relativement disjoints, comme pour identifier les communautés dans les réseaux sociaux [20].
3- Les arbres recouvrants minimaux (un fournisseur d'accès Internet voulant raccorder plusieurs destinations ensemble à la fibre optique au moindre coût est un exemple concret de ce cas de figure [21]).
4- La correspondance des graphes bipartis, dont les nœuds peuvent être divisés en 2 sous-ensembles disjoints, qui consiste à faire correspondre des nœuds d'un sous-ensemble à ceux de l'autre, comme pour un site de recherche d'emploi.
5- Le problème du flux maximum qui consiste à calculer le trafic maximal théorique entre deux nœuds uniques d'un graphe orienté (exemple des transporteurs ou des compagnies aériennes).
6- L'identification de nœuds particuliers, un ensemble de nœuds importants pouvant aider à localiser un événement (comme les cellules terroristes, la propagation d'une maladie ou les campagnes Marketing pour un produit).

2.5.2 La représentation

Un graphe peut être représenté dans une matrice d'adjacence. Un graphe de « n » nœuds sera donc représenté comme une matrice carrée « n x n ». Les arcs entre les nœuds apparaissent dans la matrice sous forme de 0, 1 où le poids en cas d'arcs pondérés. Un graphe devient creux, dit Sparse, quand le nombre d'arcs actuels est nettement inférieur à celui des arcs possibles. Par exemple dans un réseau social de « n » utilisateurs contenant « n x (n-1) » relations possibles, le nombre de relations maximum que peut avoir un utilisateur reste largement inférieur à la taille du réseau, exemple illustré dans la Figure 18. Le même principe s'applique sur les liens hypertexte dans les pages Web.

L'opposé d'un graphe creux est un arc dense, dit Tight, quand le nombre d'arcs est proche de celui des arcs possibles.

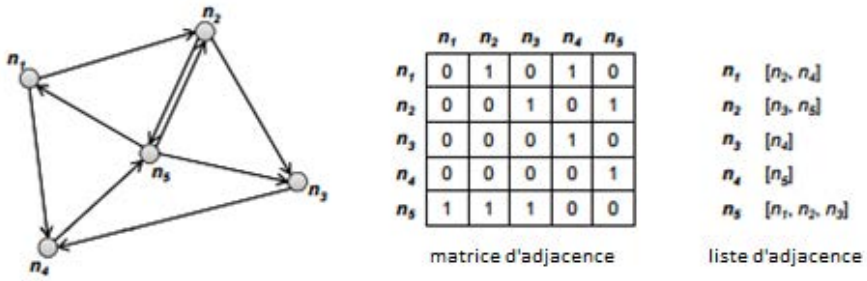

Figure 18 : Graphe sous forme de matrices et de listes d'adjacence

2.5.3 La recherche initiale parallèle

La recherche du plus court chemin à partir de la source vers tous les autres nœuds est l'un des problèmes classiques en théorie des graphes. L'algorithme de Dijkstra représenté en Figure 19, permet de résoudre ce problème par un traitement séquentiel, contrairement à MapReduce fournissant une solution par un traitement parallèle. Cette solution est l'algorithme de recherche initiale en largeur, dit Parallel Breadth-First Search.

La recherche initiale en largeur est un algorithme itératif où chaque itération correspond à une tâche MapReduce. Au premier lancement il découvre les nœuds connectés à la source; au deuxième, les nœuds connectés à ces derniers et ainsi de suite. Chaque itération permet d'élargir le périmètre de couverture. Ainsi tous les nœuds seront découverts avec leurs chemins les plus courts.

Hadoop fournit une API d'outils appelés Counters et permettant, comme leur nom l'indique, de compter les événements qui se produisent. A la fin de chaque itération, le programme pilote accède à la valeur de ces compteurs et détermine la nécessité de lancer une nouvelle itération.

Curieusement, très peu d'itérations sont nécessaires pour calculer la distance la plus courte à tous les nœuds, par conséquent, pour les problèmes rencontrés dans la vie quotidienne. Par exemple, six degrés de séparation suffisent pour connecter tous les hommes de la planète entre eux. En d'autres termes, l'exécution de l'algorithme de recherche initiale requiert au plus six itérations de MapReduce dans un réseau social quelconque.

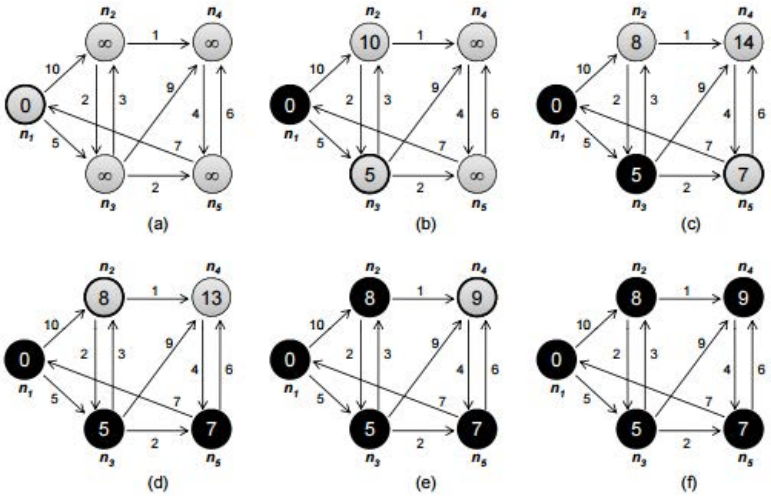

Figure 19 : Exemple d'algorithme de Dijkstra

2.5.4 L'algorithme PageRank

Il s'agit d'une mesure de qualité des pages Web reposant sur la structure d'un graphe de liens hypertexte [22]. C'est un des critères majoritaires parmi de nombreux utilisés par le moteur de recherche de Google Une valeur est attribuée à chaque page Web selon le nombre de fois que passerait par cette page un utilisateur en cliquant sur les liens qui apparaissent dans les pages. Le score de chaque page déterminera sa position dans les résultats de la recherche.

Cette mesure repose théoriquement sur la bonne foi des utilisateurs. Ceci dit, dans la vie quotidienne on constate l'influence des publicitaires sur les moteurs de recherche. Cela fausse partiellement le calcul de cette mesure en appliquant l'algorithme de recherche initiale de MapReduce, malgré les stratégies développées par les moteurs de recherche pour combattre ce phénomène. La Figure 20 et la Figure 21 illustrent l'utilisation de l'algorithme PageRank en général et son utilisation avec MapReduce en particulier.

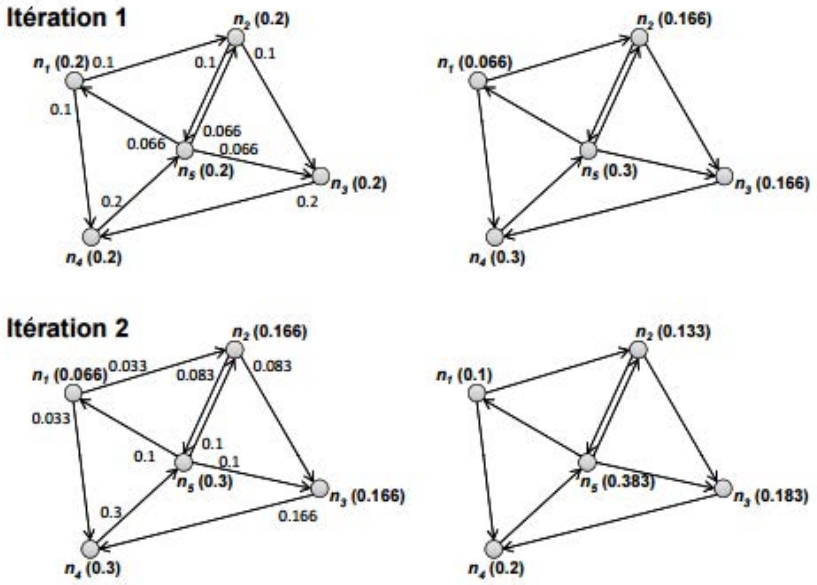

Figure 20 : Algorithme PR

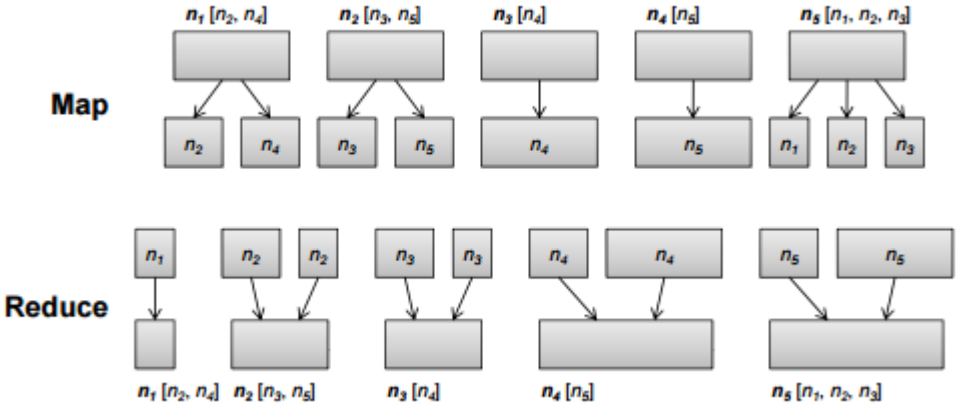

Figure 21 : Algorithme PR avec MapReduce

2.5.5 Les problèmes rencontrés

Pendant le traitement des graphes un calcul local est effectué dans chaque nœud et le résultat est transmis au nœud voisin. Il faudrait donc plusieurs itérations pour parcourir le graphe en entier. Pour les graphes les plus denses, le temps de traitement est déterminé par la quantité des données transmise sur le réseau, proportionnelle au nombre de nœuds dans le graphe. Cela rend MapReduce pratiquement impuissant quand il s'agit de traiter les graphes à forte densité.

2.5.6 Conclusion

Dans ce paragraphe on a abordé le concept de traitement des graphes avec MapReduce. On a également détaillé la technique de recherche initiale parallèle permettant de résoudre le problème de la recherche du plus court chemin à partir de la source vers tous les autres nœuds. Enfin on a abordé la technique PageRank et son utilisation dans les moteurs de recherche.

2.6 Les algorithmes EM de traitement de texte

Certaines techniques mathématiques, notamment les modèles statistiques, permettent de réaliser un traitement modélisé des données. La modélisation statistique est contrôlée par trois facteurs :

1- La sélection du modèle.
2- L'estimation des paramètres. En d'autres termes, l'application d'un algorithme d'optimisation des performances du modèle.
3- Le décodage à la sortie.

L'algorithme EM, pour Expectation Maximization, est une classe d'algorithmes qui permettent de trouver le maximum de vraisemblance des paramètres de modèles probabilistes, lorsque le modèle dépend des variables latentes non observables (données incomplètes) [23].

Cet algorithme comporte une étape d'évaluation de l'espérance « E », où l'on calcule l'espérance de la vraisemblance en tenant compte des dernières variables observées et une étape de maximisation « M », où l'on estime le maximum de vraisemblance des paramètres en maximisant la vraisemblance trouvée à l'étape « E ».

2.6.1 L'estimation de vraisemblance maximale

L'estimation de vraisemblance maximale, dite Maximum Likelihood Estimation, est une méthode statistique permettant d'inférer les paramètres de la distribution de probabilité d'un échantillon donné. Par exemple, pour un jeu de billes, prédire le comportement et le trou où tombera la bille. Pour ce faire, le modèle Bernoulli fournit une approximation du processus physique. Les performances du traitement sont alors proportionnelles à la quantité des données en question.

2.6.2 Les variables latentes

Les variables latentes, dits Latent Variables, pouvant décrire les structures linguistiques des variables observées, sont très utiles pour le traitement de textes. Elles servent à 3 choses :

1- Synthétiser les données en cours de traitement.
2- S'approcher des concepts théoriques sous-jacents à ce qui a été mesuré.
3- Comparer la théorie à l'échantillon.

2.6.3 Le modèle HMM

Le modèle HMM pour Hidden Markov Model est un modèle statistique dans lequel le système modélisé est supposé être un processus Markovien de paramètres inconnus (permettant la prédiction du futur en fonction des éléments du passé).

Ce modèle est utilisé dans applications diverses et variées, comme la reconnaissance vocale [24], l'extraction des données [25], la recherche des gênes [26], le balisage partiel des discours [27], l'estimation des stocks du marché [28], la récupération de texte [29] et l'alignement des phrases dans la traduction de texte [30].

L'apprentissage machine dans ce modèle se fait en plusieurs sessions en se basant sur les algorithmes suivants :

1- L'algorithme progressif dit Forward Algorithm qui permet de calculer la probabilité d'un état dans un certain temps en se basant sur l'historique.
2- L'algorithme de Viterbi qui permet de corriger dans une certaine mesure les erreurs survenues lors d'une transmission de données à travers un canal non fiable pendant le traitement du texte.
3- L'algorithme rétrogressif dit Backward Algorithm, qui, contrairement aux deux algorithmes précédents, agit dans le sens opposé comme son nom l'indique et ne se base pas sur les probabilités de départ.

L'utilisation des algorithmes progressif et rétrogressif est illustrée dans la Figure 22.

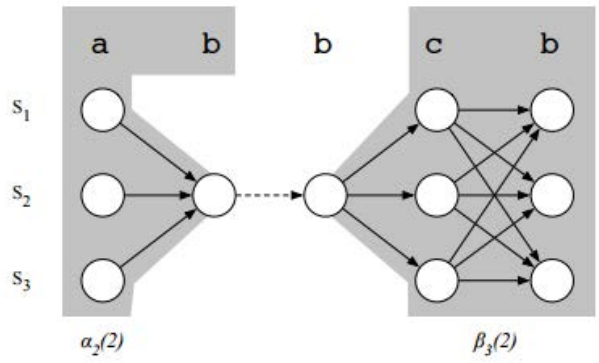

Figure 22 : Utilisation des algorithmes progressif et rétrogressif

2.6.4 L'application dans MapReduce

Chaque itération des algorithmes EM correspond à une tâche MapReduce. Toutes les tâches sont contrôlées par le programme pilote, permettant de tracer le nombre d'itérations et les critères de convergence. Le modèle de paramètres choisi ne change pas pendant toute la durée d'exécution :

1- Les « Mappers » agissent dans des sessions d'entrainement en calculant les variables latentes.
2- Les « Reducers » collectent les statistiques et agissent sur les problématiques d'optimisation.
3- Les « Combiners », en synthétisant les statistiques, permettent de réduire la quantité des données à écrire sur le disque.

Le parallélisme avec ces modèles est efficace pour plusieurs raisons :

1- Du fait que les algorithmes progressif et rétrogressif sont à l'origine des efforts de calculs, toutes les ressources du Cluster sont alors consacrées.
2- Pouvoir lancer plusieurs tâches « reduce » permet d'éviter une surcharge de données sur le réseau et une dégradation des performances.

2.6.5 La traduction automatique statistique

A travers MapReduce, la traduction automatique statistique montre l'efficacité d'un traitement en se basant sur les données. Il s'agit d'une approche dirigée par les données, dite Data-Driven Approach. Le rôle du modèle choisi est donc de sélectionner une traduction grammaticalement correcte, parmi une large variété d'hypothèses. Par conséquent, la qualité de la traduction est proportionnelle à la quantité des données soumises à l'apprentissage. Google Translate est un exemple concret de cette implémentation, illustrée dans la Figure 23.

L'alignement correct des mots nécessaire pour construire des modèles de traduction efficaces peut facilement être acquis en utilisant les algorithmes EM. Une implémentation Open Source basée sur Hadoop existe et fournit l'alignement des mots, l'extraction et la notation des phrases. Il existe également d'autres algorithmes similaires à EM et permettant de résoudre des problèmes généraux, comme Gradient-Based Optimization et Log-Linear Model.

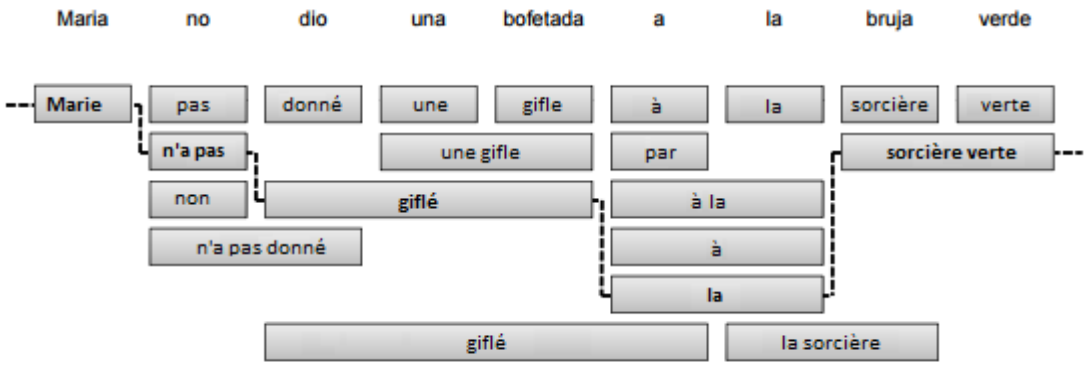

Figure 23 : Traduction automatique statistique avec MapReduce

2.6.6 Conclusion

L'algorithme EM est un algorithme itératif utilisé pour la classification des données, l'apprentissage automatique ou la vision artificielle. Cet algorithme comporte 2 étapes :

1- Une étape d'évaluation de l'espérance, où l'on calcule l'espérance de la vraisemblance en tenant compte des dernières variables observées.
2- Une étape de maximisation, où l'on estime le maximum de vraisemblance des paramètres, en maximisant la vraisemblance trouvée à la première étape.

La traduction automatique statistique utilise l'algorithme EM via un apprentissage à partir de corpus parallèles alignés. Elle ne nécessite pas l'utilisation de connaissances linguistiques liées à une langue. Elle peut s'appliquer à n'importe quelle paire de langues. Cette technique envisage la création de plusieurs alternatives ou hypothèses scorées permettant de choisir la meilleure solution.

Cependant, cette technique présente deux inconvénients :

1- Elle est considérée comme une boîte noire sans lien avec la linguistique.
1- Elle nécessite une grande masse de données d'apprentissage.

2.7 La nouvelle génération de MapReduce

La nouvelle génération de MapReduce, dite YARN, a vu le jour en 2012 dans le cadre du projet Apache Hadoop. YARN est considérée comme un socle architectural de Hadoop permettant de lancer plusieurs moteurs de traitement des données et ouvrant la voie à la séparation de la problématique de la gestion des ressources des Clusters, de celle du traitement des données. Cette séparation rend possible tout un ensemble de nouveaux projets, comme Stinger et Tez permettant, dans le cadre de certains scénarii, d'atteindre des temps de réponse adaptés à l'interaction humaine. Egalement Apache Storm s'appliquant au traitement de flux de données. Pivotal Software, le fondateur de Spring Framework, le projet permettant de construire et de définir l'infrastructure d'une application Java, a déjà annoncé le Spring YARN Framework qui s'adresse aux développeurs Java souhaitant écrire leurs propres applications YARN. Ainsi, en s'appuyant sur le système de stockage et la plate-forme de gestion des Clusters Hadoop, les applications de traitement des données peuvent offrir à leurs utilisateurs la possibilité d'interagir avec les données de multiples façons. La Figure 24 illustre une comparaison entre l'architecture Hadoop 1.0 avec MapReduce et l'architecture Hadoop 2.0 avec YARN.

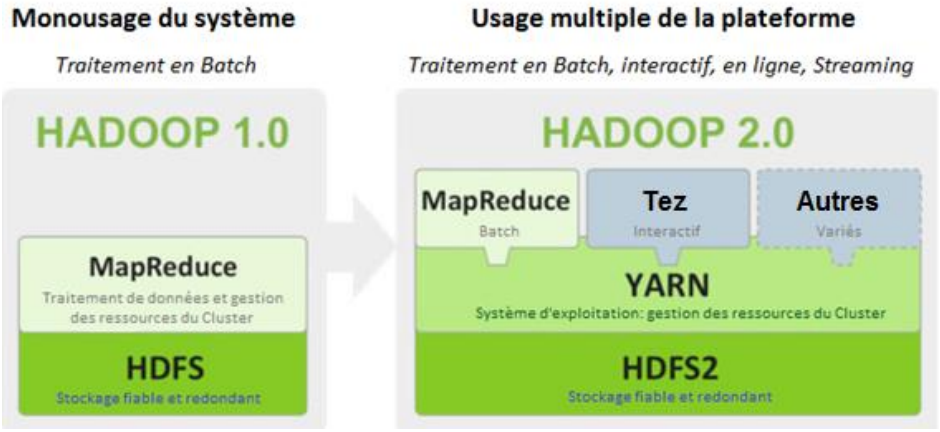

Figure 24 : Comparaison entre l'architecture Hadoop 1.0 et l'architecture Hadoop 2.0

2.7.1 Les avantages de YARN

L'architecture centrale de YARN a optimisé le fonctionnement du Cluster Hadoop de plusieurs façons.

2.7.1.1 L'architecture multi-entité

YARN permet à une multitude de moteurs de traitement (Open Source ou propriétaires) d'utiliser Hadoop comme étant un standard commun pour les traitements en Batch, l'utilisation interactive et le flux en temps-réel, dit Streaming, ayant tous accès simultanément au même stockage de données. Cette caractéristique est très importante d'un point de vue investissement des entreprises.

2.7.1.2 L'utilisation du Cluster

L'allocation dynamique des ressources du Cluster pratiquée par YARN améliore significativement l'utilisation du Cluster par rapport aux versions précédentes de MapReduce.

2.7.1.3 L'évolutivité

La puissance du traitement des données s'amplifie avec YARN puisque son gestionnaire de ressources, dit ResourceManager, se focalise exclusivement sur la planification et suit le rythme des Clusters en expansion continue, permettant de gérer des PO de données.

2.7.1.4 La compatibilité

Toutes les applications développées sur les versions précédentes de MapReduce sont inconditionnellement compatibles avec YARN.

2.7.2 Conclusion

YARN potentialise Hadoop en permettant d'intégrer de nouveaux paradigmes plus adaptés à certains usages que MapReduce :

1- Le traitement de flux de données massif.
2- Le traitement des graphes.
3- Le traitement In-Memory.

Même si des applications correspondant à ces besoins existaient pour la plupart avant l'apparition de YARN, elles devaient le plus souvent s'appuyer sur leurs propres mécanismes de gestion de distribution et d'accès aux données à travers le Cluster. YARN vient apporter une touche de simplification dans le déploiement de ces applications, effectué directement au sein du Cluster.

2.8 Apache Storm

Il s'agit d'un projet dérivé de Hadoop fournissant une solution de traitement des flux en temps-réel capable de couvrir un large éventail de cas d'emploi, y compris les analyses temps-réel, l'apprentissage automatique, les calculs continus et d'autres. Storm dispose également de capacités de traitement rapide, de façon à pouvoir traiter un million d'enregistrement de données par seconde et par nœud au sein d'un Cluster de taille moyenne.

2.8.1 La puissante combinaison de Storm et de YARN

YARN a changé fondamentalement la gestion des ressources et a activé le déploiement de nouveaux services dans l'infrastructure Hadoop. Storm profite complètement de ces avantages pour se doter de fonctionnalités plus puissantes. La Figure 25 montre la place de Storm dans l'architecture classique d'une plate-forme Hadoop basée sur YARN.

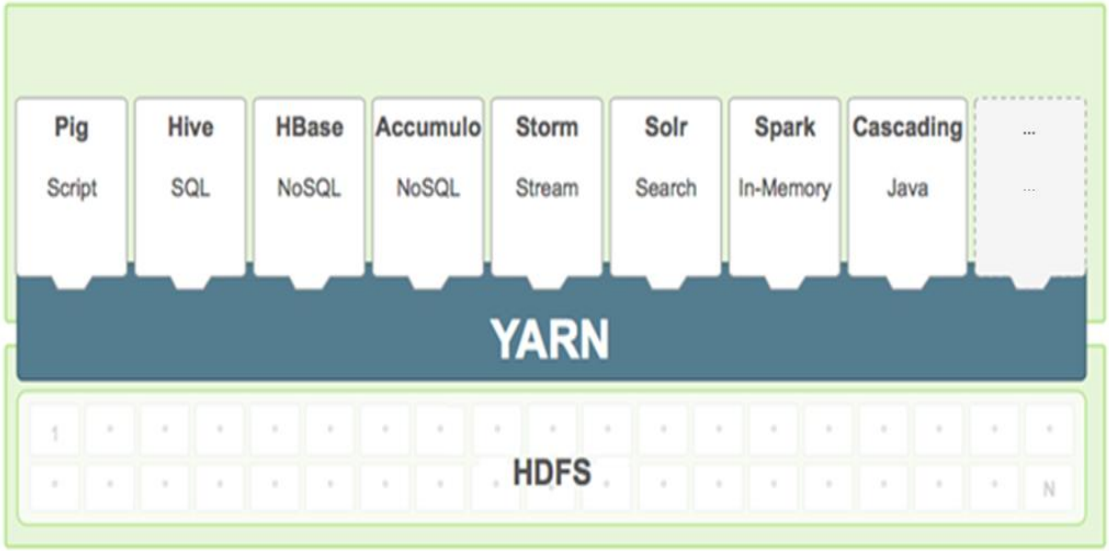

Figure 25 : Combinaison d'Apache Storm et de YARN

A terme une intégration plus profonde est prévue entre Storm et YARN afin de mieux gérer l'allocation des ressources d'un Cluster basé sur ces deux composants. Yahoo héberge un des plus grands déploiements de Storm, ce qui l'a amené à prendre en charge l'amélioration des fonctionnalités assurant sa sécurité. Il est prévu que la nouvelle version de Storm inclue donc les améliorations suivantes :

1- L'authentification Kerberos, avec des informations d'identification poussées et renouvelées automatiquement.
2- La planification multi-entité.
3- L'intégration sécurisée avec d'autres projets Hadoop, tels que ZooKeeper, HDFS, HBase et Hive.
4- L'isolation utilisateur, avec des lancements des topologies Storm à la demande.

2.8.2 Les limitations de Storm

Storm ne s'exécute pas nativement sur un Cluster Hadoop mais dépend d'autres couches, notamment ZooKeeper, lui permettant de coordonner les processus. Actuellement, Yahoo et Hortonworks développent une nouvelle version de Storm lui permettant de s'exécuter nativement à travers les fonctionnalités de YARN.

Par ailleurs, Storm est suffisamment rapide et évolutif pour que la plupart des cas d'emploi ne nécessitent que des Clusters de taille inférieure à 20 nœuds. Cependant et dans certains cas, Storm est confronté à certains problèmes sur des Clusters de plusieurs milliers de nœuds.

Enfin, d'un point de vue persistance et résilience, Storm n'est pas la solution idéale. Si un nœud tombe en panne alors que les données sont traitées en mémoire, il n'y a pas de recouvrement possible après réaffectation d'un autre nœud.

2.8.3 Storm Trident

Trident est une extension de Storm développée par Twitter et permettant de fournir une abstraction haut-niveau tout au long des traitements de flux en temps-réel avec des requêtes distribuées. Trident dispose d'une API permettant de lancer à travers un flux en entrée, un ensemble d'opérations, telles que le filtre, l'agrégation et le groupement.

Comme dans beaucoup de cas d'emploi, si le besoin est de traiter une requête pour une fois, cela peut être réalisé avec une topologie Trident. Par ailleurs, il serait compliqué de traiter des cas d'emploi de haute performance, parce qu'il ajoutera une complexité inutile.

2.8.4 Conclusion

Bien qu'assez jeune, Storm est un produit permettant de répondre à un large éventail de besoins utilisateur. Le déploiement d'un Cluster Apache Storm est relativement rapide et aisé comparé à d'autres solutions BigData. Cependant, l'usage de ce dernier n'est pas simple pour autant. En tant que système distribué, le développement sur Storm reste un exercice qui requiert de bonnes connaissances sur les particularités de ce type de programmation. Sans cela, les performances peuvent rapidement s'écrouler.

Storm contribue aujourd'hui à baisser le coût d'accès aux technologies de calcul distribué et en simplifier l'utilisation. Cela va probablement permettre de faciliter un peu plus la migration d'un certain nombre d'architectures basées sur des Batch complexes, vers des systèmes de traitement distribué de type Hadoop.

2.9 Apache Spark

Apache Spark est un Framework Open Source de traitements Big Data construit à la base de Hadoop MapReduce pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d'utilisation. Celui-ci a originellement été développé par l'Université UC Berkeley en 2009 et passé Open Source sous forme de projet Apache en 2010.

2.9.1 L'écosystème de Spark

À côté des API principales de Spark, l'écosystème contient des librairies additionnelles qui permettent de travailler dans le domaine des analyses BigData et de l'apprentissage automatique. Parmi ces librairies, on trouve Streaming, SQL, MLlib et GraphX.

2.9.1.1 Spark Streaming

Il peut être utilisé pour des traitements de flux en temps-réel. Il s'appuie sur un mode de traitement en micro-Batch et utilise une abstraction permettant de réutiliser efficacement les données dans une large famille d'applications appelée RDD pour Resilient Distributed Dataset. Les RDD sont tolérants à la panne et proposent des structures de données parallèles qui permettent aux utilisateurs de :

1- Persister explicitement les données intermédiaires en mémoire.
2- Contrôler leur partitionnement afin d'optimiser l'emplacement des données.
3- Manipuler les données, en utilisant un ensemble important d'opérateurs.

2.9.1.2 Spark SQL

Elle permet d'exposer les jeux de données Spark, via une API de type JDBC et d'exécuter des requêtes de type SQL, en utilisant les outils BI et de visualisation traditionnels. Spark SQL permet d'extraire, de transformer et de charger des données sous différents formats et les exposer pour des requêtes ad hoc.

2.9.1.3 Spark MLlib

MLlib est une librarie d'apprentissage automatique, qui contient tous les algorithmes et utilitaires d'apprentissage classiques, comme la classification, la régression, le Clustering, le filtrage collaboratif, la réduction de dimensions, en plus des primitives d'optimisation sous-jacentes.

2.9.1.4 Spark GraphX

GraphX est la nouvelle API encore en version alpha, pour les traitements de graphes et de parallélisations de graphes. GraphX étend les RDD de Spark en introduisant le Resilient Distributed Dataset Graph, un multi-graphe orienté avec des propriétés attachées aux nœuds et aux arcs. Pour le support de ces traitements, GraphX expose un jeu d'opérateurs de base, tels que Subgraph, JoinVertices et AggregateMessages. De plus, GraphX inclut une collection toujours plus importante d'algorithmes permettant de simplifier les tâches d'analyse de graphes.

2.9.2 Les avantages de Spark

Spark présente plusieurs avantages par rapport aux autres technologies comme Hadoop et Storm. D'abord, Spark propose un Framework complet et unifié pour répondre aux besoins de traitements BigData pour divers jeux de données, différents par leur nature (texte, graphe) aussi bien que par le type de source (en mode Batch ou en temps-réel). Ensuite, Spark permet à des applications sur des Clusters Hadoop d'être exécutées jusqu'à 100 fois plus vite en mémoire, 10 fois plus vite sur le disque. Il permet d'écrire rapidement des applications en Java, Scala ou Python et inclut un jeu de plus de 80 opérateurs haut-niveau. De plus, il est possible de l'utiliser de façon interactive pour requêter les données depuis une fenêtre de commande Shell.

Enfin, en plus des fonctions « map » et « reduce », Spark supporte les requêtes SQL et le flux de données et propose des fonctionnalités d'apprentissage automatique, dit Machine Learning et de traitements orientés graphe. Les développeurs peuvent utiliser ces possibilités séparément ou en les combinant en une chaîne de traitement complexe.

2.9.2.1 Les traitements micro-Batch

Grâce à l'arrivée de Spark destiné au Streaming, le concept des micro-Batchs a suscité l'attention des développeurs. Cela consiste à morceler en traitements plus petits, des flux en temps-réel à traiter, à des intervalles entre 500 ms et 5000 ms.

Spark implémente le concept des micro-Batchs dans son fonctionnement. En revanche, Spark ne doit pas être considéré, comme un moteur de traitement de flux en temps-réel. C'est en effet la plus grande différence entre Spark et Storm.

Storm d'ailleurs, grâce à son API Trident, est en mesure de supporter les traitements Batch Standards, ainsi que les traitements micro-Batch.

2.9.3 Les limitations de Spark

Malgré tous les avantages de Spark par rapport à Hadoop, il reste confronté à une série de limitations qui empêche son accès à davantage de maturité.

2.9.3.1 Les problèmes de gestion de mémoire

Les paramètres de configuration par défaut de Spark sont très contraignants. Par exemple, le paramètre « spark.driver.maxResultSize » limité à 1 GO ne permet pas de faire des opérations « reduceByKey » sur des stockages de données de plus de 1.8 TO. Par ailleurs, le paramètre « spark.akka.frameSize » limité à 128 MO réduit significativement le nombre de tâches lancées en parallèle.

Ces problèmes sont bien entendu gérables dans les fichiers de configuration. Mais ceci demande une connaissance technique supplémentaire loin d'être disponible chez les utilisateurs standards.

2.9.3.2 Le problème des fichiers de petite taille

Spark comme Hadoop ont du mal avec les fichiers de petite taille. Raison pour laquelle on introduit la technique de dé-normalisation des données. Mais le fait est que les données sont stockées sur les nœuds en format GZIP. En partant du principe que le format de compression impose

la contrainte de disponibilité de tous les morceaux d'un fichier sur un seul nœud pour pouvoir procéder à la décompression, cela signifie un effort supplémentaire pour gérer les données sur ce nœud et une opposition au principe de traitement décentralisé de MapReduce. Pour gérer ce problème, des opérations très coûteuses de type Shuffle sont nécessaires au sein du Cluster.

La solution définitive serait de gérer la compression des données avec un format découpable, tel que LZO, plus compatible avec le traitement décentralisé des données. De plus, il faudrait s'interroger sur l'opportunité d'augmenter la taille de chaque morceau tout en diminuant leur nombre.

2.9.3.3 Le Streaming

Dans toutes les conférences de présentation de Spark, son module Streaming est mis en avant comme étant une solution simple et facile à gérer. Ceci est peut-être vrai dans un périmètre très limité. Malheureusement, dans la réalité il est très compliqué de mettre en place un tunnel de traitement de flux en temps-réel qui tient 24 heures sur 24 et 7 jours sur 7. Les correctifs Spark améliorent cette fonctionnalité petit à petit mais on est loin encore d'une solution à la fois efficace et conviviale. Pour le moment, il est préférable d'adopter Storm comme solution pour cet usage spécifique.

2.9.3.4 Le langage Python

En utilisant Spark, il est préférable en général de développer les applications avec Scala et Java, plutôt que Python. La raison est tout simplement que les nouveautés des versions récentes de Spark se trouvent, plutôt dans l'un, que dans l'autre. Alors pour Python API, les mises-à-jour complémentaires arrivent après. Résultat, Scala et Java prennent de l'avance.

Par ailleurs, dans une approche RDD, les applications basées sur du Python semblent être moins performantes que leurs homologues en Scala ou en Java.

2.9.3.5 Les erreurs aléatoires

Finalement et d'un point de vue général, l'utilisation de Spark n'est pas encore à l'abri de quelques erreurs aléatoires et très techniques. Une raison supplémentaire pour laquelle Spark ne peut pas être considéré comme une solution conviviale, pour des utilisateurs non informaticiens.

2.9.4 Conclusion

Dans ce paragraphe on a vu les caractéristiques d'Apache Spark en matière de traitement et d'analyse de données. Spark se positionne par rapport aux implémentations MapReduce traditionnelles comme Apache Hadoop. Il s'appuie sur le même système de stockage de fichiers qu'Hadoop, il est donc possible d'utiliser Spark et Hadoop ensemble dans le cas où des investissements significatifs ont déjà été faits avec Hadoop.

Il est possible également de combiner les types de traitements Spark avec Spark SQL, Spark Machine Learning et Spark Streaming, grâce à différents modes d'intégration et adaptateurs Spark. Cependant, Spark n'est pas un écosystème encore complètement mature. Il nécessite encore des améliorations dans certains domaines comme la sécurité ou l'intégration avec des outils d'analyse décisionnelle.

2.10 Conclusion du chapitre

Le traitement des données volumineuses BigData n'a jamais été aussi simple. Aujourd'hui MapReduce permet de résoudre des problèmes à des échelles encore inimaginables il y a quelques années. Mais comme tout autre outil, il reste limité.

2.10.1 Les limitations de MapReduce

Certains algorithmes dépendent d'un état global partagé pendant le traitement, ce qui rend leur implémentation difficile dans MapReduce. Une des solutions serait de construire un magasin de données distribuées capable de maintenir l'état global. Cependant, un tel système devrait être hautement évolutif pour être utilisé en conjonction avec MapReduce. Google BigTable en est un exemple, idem pour Amazon Dynamo [31]. Une solution Open Source avec HBase ou Cassandra reste encore prématurée.

2.10.2 Les solutions alternatives

Les algorithmes de Streaming représentent un modèle alternatif de programmation pour traiter des données volumineuses avec des ressources limitées de stockage et de calcul [32]. Dans ce contexte de traitement de texte, les algorithmes de Streaming ont été appliqués aux langages de modélisation [33], à la traduction par modélisation [34] et la détection de dépêches dans un flux d'information [35].

L'idée d'implémenter les algorithmes de Streaming a été généralisée dans le Framework de Microsoft Dryad [36] [37] (graphes en flux de données aléatoires) et Google Pregel, utilisant un modèle de programmation inspiré par le modèle BSP de Valiant [38] [39]. Il existe aussi une approche dans Hadoop qui consiste à ajouter une troisième étape « merge » après « map » et « reduce » fournissant un meilleur support pour les opérations relationnelles [40] :

1- Google Pig est une plate-forme de données analytiques fournissant un pseudo-langage pour manipuler les données volumineuses.
2- Hadoop Hive permet d'interroger des données relationnelles volumineuses stockées dans HDFS via des requêtes SQL [41].

2.10.3 Au-delàs de MapReduce

La problématique de traitement des données volumineuses ne cessera d'augmenter avec le temps. L'évolution technologique dans l'industrie des serveurs et les innovations dans le monde logiciel ont fait en sorte que les outils de traitement à grande échelle soient à la portée de tous. Les idées suivantes ont ouvert la voie à des évolutions gigantesques dans les algorithmes de traitement :

1- Le Scale Out ou l'ajout d'un serveur, au lieu d'augmenter les ressources d'une machine.
2- La relocalisation du code d'exécution, plutôt que de déplacer les données.
3- L'isolation de l'environnement de développement de la gestion bas niveau du système.

Google a fait un grand pas avec cette innovation, de même pour Yahoo qui a réalisé la version Open Source appelée Hadoop et rendue accessible à tous.

2.10.4 Tableau comparatif des technologies Hadoop

Ce chapitre a décrit les deux projets Spark et Storm en termes d'avantages par rapport à la version standard de Hadoop. Le Tableau 1 présente une comparaison des différentes technologies expliquées et leur usage. La couleur jaune signifie une présence possible et la couleur verte met en avant les bonnes performances d'une technologie.

Fonctionnalité	**Storm Trident**	**Spark Streaming**
Langages de programmation	Java, Scala, Python	Java, Scala, Python
Fiabilité	Autres modes de traitement inférieur, égal ou supérieur à 1 fois	Traitement en 1 fois
Flux de données	Tuple, Partition	HDFS
Persistance	MapState	RDD
Gestion d'état	Pris en charge	Pris en charge
Gestion des ressources	YARN	YARN
Approvisionnement	Surveillance avancée	Surveillance basique
Faible latence	Meilleure latence avec moins de restrictions	Plus de restrictions
Coûts de développement	Pas de réutilisation de code possible	Possibilité de réutilisation de code entre le traitement des flux et le celui en mode Batch
Garantie de livraison des messages	Possible pour les traitements en 1 fois	Possible pour les traitements en 1 fois
Persistance aux pannes	Pris en charge	Pris en charge

Tableau 1 : Comparaison des différentes technologies Hadoop

Chapitre 3. Les recherches précédentes et les motivations de l'approche de la modélisation intégratrice

3.1 Introduction au chapitre

Dans le chapitre précédent on a décrit le modèle de programmation MapReduce, ses aspects et son impact dans notre quotidien. On a abordé les concepts de base de plusieurs types de traitement tels que les graphes, les index et les données textuelles, avec des exemples de la vie quotidienne. On a également fourni une description des principaux projets appartenant à l'univers de Hadoop MapReduce, tels que YARN, Apache Storm et Apache Spark.

Dans les paragraphes qui suivent, on va présenter concrètement les techniques et les patrons de modélisation. Mais quelques généralités sont à prendre en considération :

1- Tout d'abord, les modèles de données non-relationnels sont souvent initiés par les requêtes applicatives, contrairement à la modélisation relationnelle.
2- La modélisation des données NoSQL, nécessite une profonde compréhension des structures de données et des algorithmes, plus que de la modélisation relationnelle. On décrira par la suite plusieurs structures de données connues, non spécifiques au NoSQL mais très utiles pour y pratiquer la modélisation.
3- La duplication des données et la dé-normalisation sont considérées comme deux techniques incontournables.
4- Les bases de données relationnelles ne sont pas très pratiques pour la modélisation hiérarchique ou encore le traitement et la modélisation de type graphe. Les bases de données orientées graphe sont évidemment une solution parfaite dans ce domaine mais la plupart des solutions NoSQL sont capables de résoudre ces problèmes.

3.2 Les techniques de modélisation

Les outils d'acquisition des données et les techniques de modélisation des données permettent de faire une translation de la conception des systèmes complexes vers des représentations compréhensibles des flux de données et des processus de calcul, en mettant en vigueur un plan de reconstruction et d'ingénierie. Les modélisateurs de données utilisent souvent des modèles multiples pour représenter les mêmes données et s'assurer que tous les processus, entités, liaisons et flux de données, ont été identifiés. Il existe donc des approches diverses et variées de la modélisation des données. On distingue trois grandes familles :

1- La famille des techniques de modélisation conceptuelle.
2- La famille des techniques de modélisation générale.
3- La famille des techniques de modélisation hiérarchique.

3.2.1 La modélisation conceptuelle

Ce paragraphe est consacré aux principes de bases de la modélisation NoSQL. Les techniques conceptuelles identifient les liaisons du plus haut niveau entre des entités différentes. Dans cette famille, on distingue en général les techniques suivantes:

3.2.1.1 La dé-normalisation

La dé-normalisation peut être définie, comme une copie des mêmes données dans de multiples documents ou tables dans le but de simplifier ou optimiser le traitement des requêtes ou encore ajuster les données utilisateur dans un modèle particulier. En général, la dé-normalisation est utile dans les cas suivants :

1- Le calcul du volume de données en entrée / sortie par rapport au volume total : En utilisant la dé-normalisation, il est possible de regrouper toutes les données nécessaires pour traiter une requête dans un seul endroit. Cela signifie que pour les différents flux de requêtes, les mêmes données seront accessibles par différentes combinaisons. C'est pourquoi, on devrait dupliquer les données et augmenter le volume total.
2- Le calcul de la complexité par rapport au volume total : Une normalisation modélisation / temps et une jointure requête / temps conséquente, permettent d'augmenter la complexité du moteur de traitement des requêtes, surtout dans les systèmes distribués. La dé-normalisation permet de stocker les données dans une structure sous forme de requête et de simplifier le traitement par conséquent.

La modélisation conceptuelle s'applique dans les stockages clé-valeur, les bases de données BigTable et les bases de données orientées document.

3.2.1.2 Les agrégations

Cette technique est considérée comme la meilleure façon d'améliorer les performances dans un stockage de données volumineux. Il s'agit de fournir des enregistrements de données agrégés qui cohabitent avec les données principales dans la base. Ces enregistrements de données disposent d'un effet significatif sur les performances des requêtes.

L'utilisation des techniques d'agrégation lors de la modélisation des données, est une des méthodes de travail les plus connues, garantissant les propriétés ACID du système. Tous les principaux types de bases de données NoSQL, offrent des fonctionnalités de schémas d'abstraction d'une manière ou d'une autre :

1- Les stockages clé-valeur et les modèles orientés graphe n'imposent pas de contraintes de valeurs. Ces valeurs peuvent donc être constituées de formats arbitraires. Il est possible également de faire varier un certain nombre d'enregistrements de données pour une entité métier unique en utilisant des clés composites. Par exemple, un compte utilisateur peut être modélisé par un certain nombre d'éléments avec une clé composite, comme identifiant / nom, identifiant / email ou identifiant / messages. Si l'utilisateur ne dispose pas d'adresse email ou de messages, alors les entrées correspondantes ne sont pas sauvegardées.
2- Les modèles de données BigTable gèrent les schémas d'abstraction à travers un ensemble variable de colonnes via la famille de colonnes et plusieurs versions d'une même cellule.
3- Les bases de données orientées document ne contiennent pas de schéma explicite. Certaines permettent de valider les données en entrée, en utilisant des schémas d'abstraction prédéfinis par l'utilisateur.

Les schémas d'abstraction permettent de construire des classes d'entités à structure interne complexe (entités imbriquées) et de faire varier la structure de certaines entités particulières. Cette fonctionnalité fournit 2 avantages majeurs :

1- La réduction des relations « 1-n » à l'intermédiaire des entités imbriquées, par conséquent, la réduction des jointures.
2- Le masquage des différences techniques entre les entités métier et la modélisation des entités métier hétérogènes, en utilisant une collection de documents ou une table.

Ces facilités sont illustrées dans la Figure 26. Cette figure représente la modélisation d'une entité de produit dans le domaine d'activité e-commerce. Tout d'abord, on peut constater que tous les produits disposent d'un identifiant, un prix et un descriptif. Ensuite, on constate que les différents types de produits ont des attributs différents (Auteur pour Livre, Taille pour Pantalon). Certains sont soumis à une relation de type « 1-n » ou « n-n » (Pistes pour Album musical). De plus, il est possible que certaines entités ne puissent pas être modélisées, en utilisant des types fixes. Par exemple, les attributs de Pantalon ne sont pas cohérents, à travers les marques et sont spécifiques à chaque fabriquant.

Il est possible de surmonter tous ces aspects dans un modèle relationnel normalisé mais les solutions seraient très complexes. Le schéma d'abstraction permet d'utiliser une agrégation unique (Produit) qui peut modéliser tout type de produit et ses attributs. Une intégration avec dé-normalisation a un impact considérable sur la performance et la consistance à la fois. Par conséquent il faudrait être vigilant lors de la mise-à-jour des flux de données.

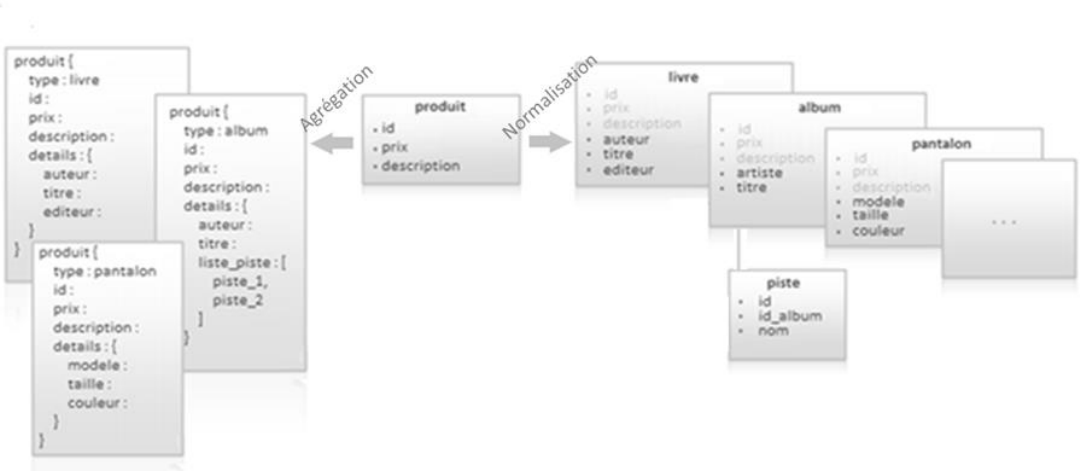

Figure 26 : Agrégation d'entités

La technique d'agrégation s'applique dans les stockages clé-valeur, les bases de données BigTable et les bases de données orientées document.

3.2.1.3 Les jointures côté application

Les jointures sont rarement supportées dans les solutions NoSQL, vu la nature orientée question des bases de données non-relationnelles. Par conséquent, les jointures sont souvent assurées en phase de conception, contrairement aux modèles de données relationnels, où les jointures sont assurées en phase d'exécution. Le temps des requêtes de jointures est souvent pénalisant pour les performances mais dans plusieurs cas de figure, il est possible d'éviter les jointures en utilisant la dé-normalisation et les agrégations (intégration des entités imbriquées). Bien sûr, dans beaucoup de cas les jointures sont inévitables et doivent être assurées par une application. Les cas d'emploi les plus importants sont :

1- Les relations « n-n », sont souvent modélisées par des liens et nécessitent des jointures.
2- Les agrégations sont souvent inapplicables, quand les entités internes subissent des modifications. Il vaut mieux appliquer généralement une jointure, pendant le temps de requête, contrairement au changement de valeurs. Par exemple, un système de messagerie peut être modélisé, en tant qu'une entité utilisateur, contenant des entités Message imbriquées. Mais si les messages sont souvent concaténés, il vaut mieux extraire ces messages en tant qu'entités indépendantes et appliquer une jointure avec l'entité Utilisateur pendant le temps de requête, comme illustré dans la Figure 27.

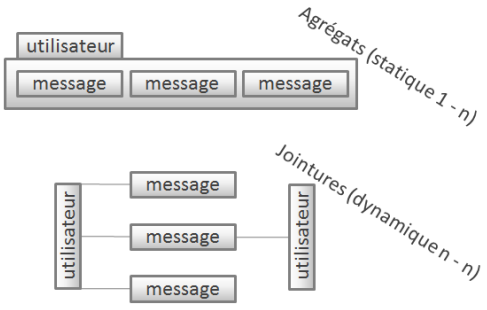

Figure 27 : Agrégation et jointures

La technique des jointures côté application s'applique dans les stockages clé-valeur, les bases de données BigTable, les bases de données orientées document et les bases de données orientées graphe.

3.2.2 La modélisation générale

Dans ce paragraphe sont abordées les techniques générales de modélisation s'appliquant sur une variété d'implémentations NoSQL. Pour la plupart, les moteurs de bases de données non-relationnelles ne disposent pas d'une interface transactionnelle développée. En revanche et dans certains cas, le comportement transactionnel des SGBD peut être réalisé différemment.

3.2.2.1 Les agrégations atomiques

La plupart des implémentations NoSQL ont un support limité des transactions. Dans certains cas il est possible d'accomplir un comportement transactionnel en utilisant des verrous distribués. Mais il est possible de modéliser les données en utilisant les techniques d'agrégation pour garantir certaines propriétés ACID. Une des raisons pour lesquels un mécanisme transactionnel puissant est une partie indissociable des bases de données relationnelles, est que les données normalisées nécessitent des mises-a-jour multiplaces. D'un autre côté, les agrégations permettent de stocker une seule entité métier, en tant que document, ligne ou couple clé-valeur et la met à jour automatiquement. Évidemment, l'agrégation atomique en tant que technique de modélisation des données n'est pas une solution transactionnelle complète mais si le stockage fournit certaines garanties d'atomicité, de verrous ou d'instructions de type Test-and-set (instructions atomiques utilisées pour écrire dans un emplacement mémoire et retourner la valeur d'origine de cet emplacement), les agrégations atomiques peuvent alors être appliquées. La Figure 28 illustre l'usage des agrégations atomiques.

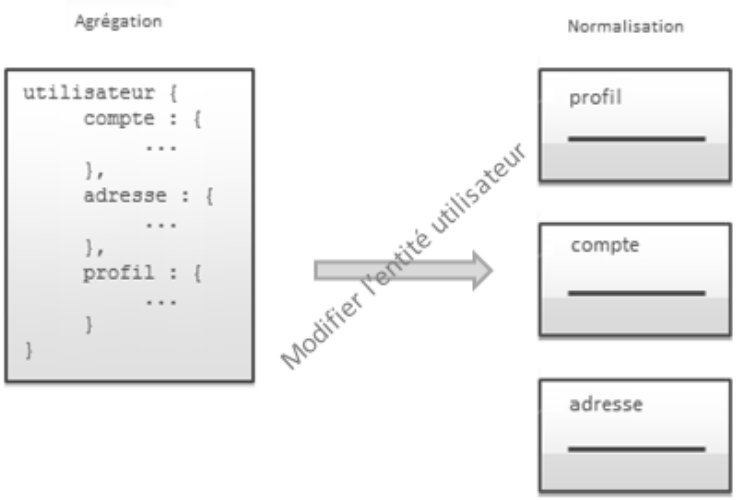

Figure 28 : Usage des agrégations atomiques

La technique des agrégations atomiques s'applique dans les stockages clé-valeur, les bases de données BigTable et les bases de données orientées document.

3.2.2.2 Les clés énumérables

Cette technique permet de morceler un modèle de données clé-valeur à travers des serveurs multiples en appliquant la méthode de hachage de la clé, comme illustré dans la Figure 29. Cette technique s'applique en conséquence du tri, rendant plus complexe la gestion des données. Les avantages les plus importants d'un modèle de données clé-valeur non trié se résument probablement au fait que les données en entrée peuvent être partitionnées à travers les serveurs multiples et simplement via un hachage de clé. Certes, le tri permet d'augmenter la complexité mais parfois, une application est capable de profiter des avantages des clés triées, même si le stockage de données n'est pas en mesure de fournir une telle fonctionnalité. Dans l'exemple précédent de modélisation de système de messagerie, on constate les faits suivants :

1- Certains stockages NoSQL fournissent des compteurs atomiques permettant de générer des identifiant séquentiels. Dans ce cas, il est possible de stocker les messages, en utilisant des clés composites de type identifiant / utilisateur ou identifiant / message. Si l'identifiant du message le plus récent est connu, il est alors possible de parcourir le message précédent. Ainsi il est possible de parcourir le message précédent ou suivant d'un message quelconque grâce à son identifiant.
2- Les messages peuvent être groupés dans des ensembles (messages du jour). Cela permet de parcourir la boîte de réception en avant et en arrière à partir de n'importe quelle date spécifique ou à partir de la date en cours.

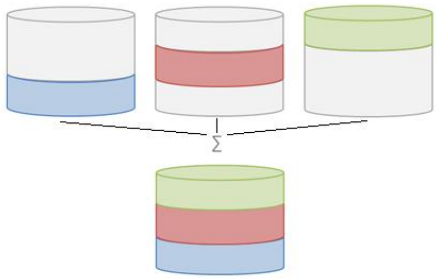

Figure 29 : Clés énumérables

Finalement la technique des clés énumérables s'applique dans les stockages clé-valeur uniquement.

3.2.2.3 La réduction dimensionnelle

La réduction dimensionnelle permet de faire la liaison entre les données multidimensionnelles et un modèle clé-valeur ou d'autres modèles simplistes non-multidimensionnels. Par exemple la méthode d'encodage Geohash (pratique de réduction dimensionnelle des coordonnées géographiques) permet de stocker les informations géographiques en utilisant des modèles de données plats tels que les couples clé-valeur triés, permettant de préserver les liaisons spatiales. Cette technique utilise un parcours en « Z » dans un espace bidimensionnel en encodant chaque mouvement en 0 ou 1 selon la direction. Ainsi seront transcrits les bits correspondant aux mouvements de longitude et latitude. Le processus d'encodage est illustré dans la Figure 30, où les bits rouges et noirs symbolisent la longitude et la latitude respectivement. Cette technique dispose également d'une fonctionnalité importante qui est la capacité d'estimer une distance entre régions, en utilisant les codes de proximité.

Les systèmes d'information géographiques traditionnels utilisent une arborescence quadridimensionnelle ou une arborescence multidimensionnelle pour les index. Ces structures nécessitent d'être mises-à-jour sur place et sont très coûteuses en manipulation dans le cas des larges volumes de données. Une approche alternative consiste à parcourir une structure bidimensionnelle et de l'aplatir en une simple liste de données d'entrée.

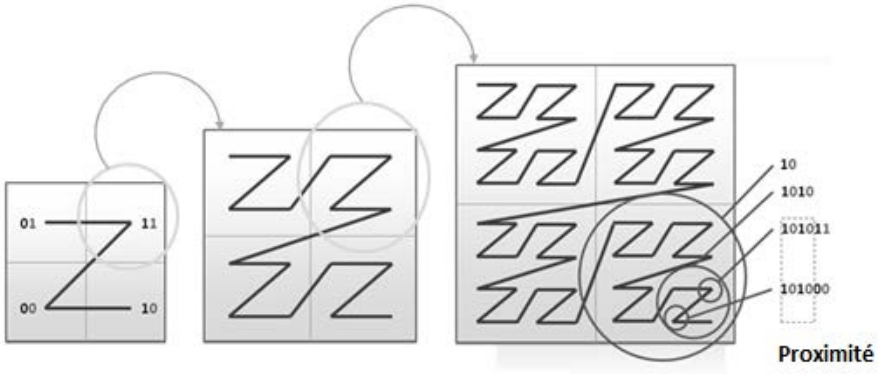

Figure 30 : Mécanisme Geohash

La réduction dimensionnelle s'applique dans les stockages clé-valeur, les bases de données BigTable et les bases de données orientées document.

3.2.2.4 Les tables d'index

C'est la technique la plus utilisée dans les bases de données BigTable. Elle consiste à créer et maintenir une table spéciale contenant des clés de liaison avec les patrons d'accès. Une table d'index peut être mise-à-jour à chaque mise-à-jour de la table maîtresse ou en mode de traitement par lot, dit mode Batch. Dans tous les cas, cela se fait au détriment des performances et crée un problème de consistance.

La technique des tables d'index est une technique simple permettant de profiter des index dans les stockages incompatibles avec les index de tables. Par exemple dans la Figure 31, il s'agit d'une table maîtresse qui stocke les comptes utilisateur accessibles par l'identifiant. Une requête permettant de récupérer tous les utilisateurs par ville peut être facilement réalisée grâce à une table supplémentaire ayant la ville comme clé d'accès.

Figure 31 : Exemple de table d'index

La technique des tables d'index s'applique dans les bases de données BigTable uniquement.

3.2.2.5 La clé d'index composite

Un index multidimensionnel peut être également créé en utilisant la technique d'indexation des clés composites. Les clés composites peuvent être utilisées non seulement dans le cadre d'une indexation mais aussi dans d'autres opérations de groupement.

La clé composite est une technique générique mais extrêmement bénéfique dans les stockages avec des tris par clé. Les clés composites avec un tri secondaire permettent de construire une sorte d'index multidimensionnel similaire à la technique de réduction dimensionnelle. Par exemple dans la Figure 32, on dispose d'un ensemble d'enregistrements de données utilisateur dont chacun correspond à une statistique. En faisant une agrégation de ces données par la région dont l'utilisateur est originaire, on peut structurer les clés par un format particulier « departement : ville : id », permettant de boucler sur les enregistrements de données d'un département ou d'une ville spécifiques, si ce stockage supporte la sélection des ensembles de clés par une correspondance de clé partielle (comme le cas des bases de données BigTable).

$SELECT\ valeur\ WHERE\ departement = 78{:}*$
$SELECT\ valeur\ WHERE\ ville = "78{:}\ chatou*"$

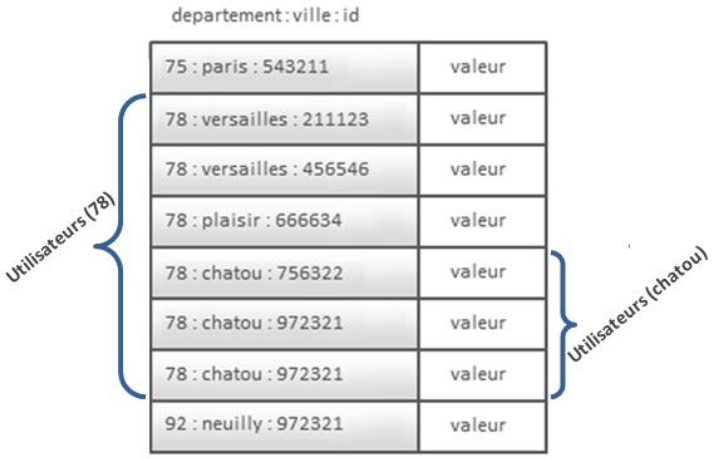

Figure 32 : Clé d'index composite

Finalement, la technique des clés composites s'applique dans les bases de données BigTable.

3.2.2.6 L'agrégation avec des clés composites

Les clés composites peuvent être utilisées non seulement pour l'indexation mais aussi pour différents types de groupement. Dans la Figure 33 on dispose d'une table d'enregistrements de données de type journal d'activité utilisateur, en termes de navigation sur les sites Internet (clics de souris). L'objectif est de compter le nombre d'utilisateurs uniques de chaque site Internet.

$$SELECT\ COUNT(DISTINCT(id))\ FROM\ clics\ GROUP\ BY\ site$$

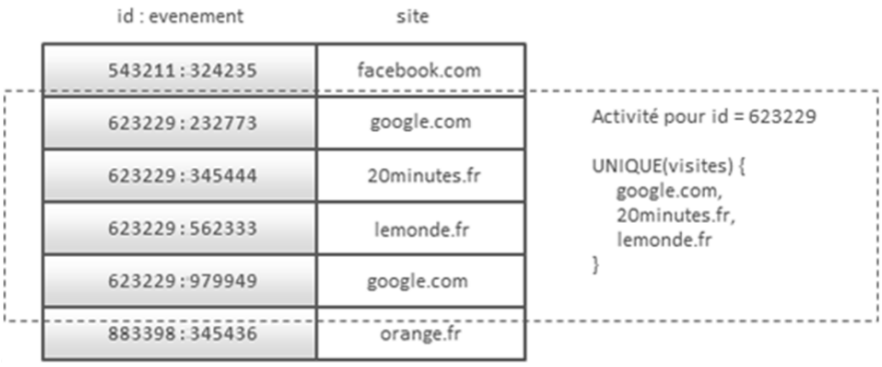

Figure 33 : Comptage d'utilisateurs uniques en utilisant des clés composites

Ce cas peut être modélisé en utilisant une clé composite avec un préfixe « id » de l'utilisateur. L'idée est de garder tous les enregistrements de données correspondant à un utilisateur unis ensemble afin de pouvoir les récupérer et les stocker en mémoire (une instance utilisateur ne génère pas beaucoup d'événements) et d'éliminer les doublons de sites Internet en utilisant les tables de hachage ou d'autres. Une technique alternative consiste à ajouter, à partir d'un seul point d'entrée relatif à un utilisateur, à la réalisation de chaque événement, tous les sites Internet à ce point d'entrée. Néanmoins, dans la plupart des implémentations, la modification d'une entrée est généralement moins efficace que dans le cas de l'insertion.

Finalement, cette technique s'applique dans les stockages clé-valeur avec tri et dans les bases de données BigTable.

3.2.2.7 La recherche inversée et l'agrégation directe

La recherche inversée est une technique davantage considérée comme patron de traitement des données, que de l'ordre de la modélisation. Néanmoins les modèles de données sont également impactés par son usage. L'idée principale de cette technique est d'utiliser un index pour trouver les données correspondant à un critère, puis d'agréger les données en utilisant la représentation d'origine ou des balayages complets. Dans l'exemple de la Figure 34, il existe un certain nombre d'enregistrements de données de type journal d'activité utilisateur sur Internet (clics de souris). Chaque enregistrement de données est constitué des éléments suivants :

1- L'identifiant.
2- La catégorie (homme, femme, bloggeur).
3- La ville d'origine.
4- Le site Internet visité.

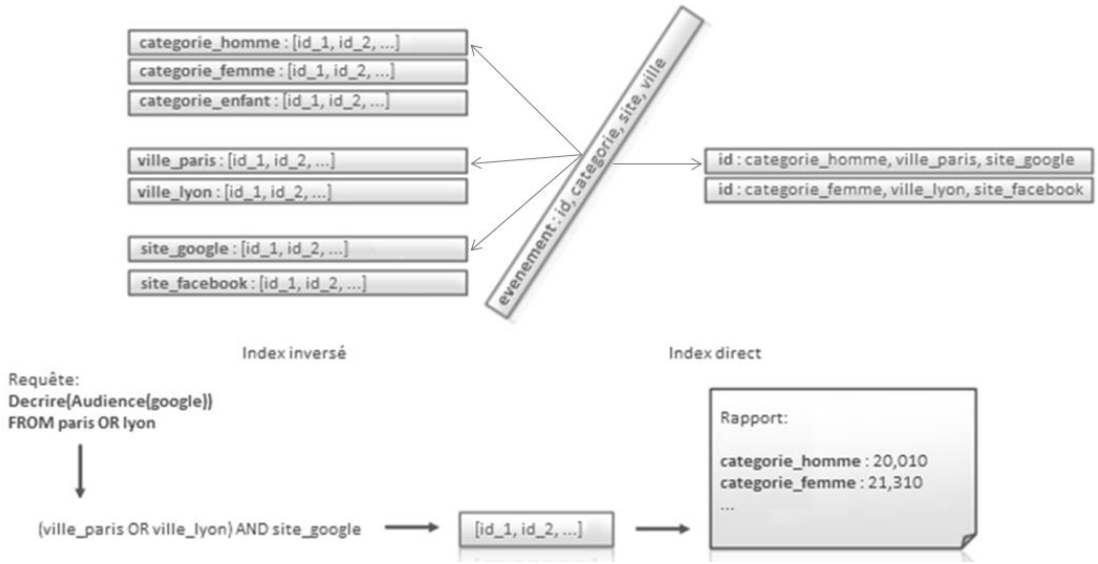

Figure 34 : Comptage d'utilisateurs uniques en utilisant un index direct ou inversé

L'objectif est de décrire l'état d'audience correspondant à un critère (site Internet ou ville), en termes de nombre d'utilisateurs uniques pour chaque catégorie. Il est clair que la recherche d'utilisateurs correspondant à un critère peut être réalisée efficacement en utilisant leurs index inversés « categorie -> [id] » ou « site -> [id] ». En utilisant de tels index, il est possible de faire des opérations de type union ou intersection correspondant aux identifiants utilisateur. Cela peut être accompli d'une manière rapide et efficace si les identifiants utilisateur sont stockés dans des listes triées. Ainsi, l'audience correspondant à un critère donné est obtenue.

En revanche, la description d'audience similaire à l'agrégation dans la requête suivante, ne peut pas être supportée efficacement en utilisant des index inversés et si le nombre de catégories est important.

$$SELECT\ COUNT(DISTINCT(id))\ \ldots GROUP\ BY\ categorie$$

Pour ce faire, il faut construire un index direct de la forme « id -> [categorie] » et ensuite lancer une itération dans le but de créer un rapport final, comme indiqué dans la Figure 34. Finalement, on devrait prendre en considération que la récupération aléatoire d'enregistrements de données pour chaque identifiant utilisateur dans l'audience ne peut être efficace d'un point de vue performance. On peut contrer ce problème en mettant en place un traitement de requêtes en masse. Cela signifie que certains ensembles utilisateur (correspondant à des critères différents), peuvent être pré-calculés et que tous les rapports relatifs à ces audiences peuvent être calculés par la suite avec un balayage complet d'index direct ou inversé.

Cette technique s'applique dans les stockages clé-valeur, les bases de données BigTable et les bases de données orientées document.

3.2.3 La modélisation hiérarchique

Ce paragraphe va aborder les techniques hiérarchiques de modélisation s'appliquant sur une variété d'implémentations NoSQL. Cette technique consiste à organiser les données dans la base sous forme d'une structure en arborescence permettant de représenter les informations en utilisant des liens parent / enfant. Chaque nœud parent peut avoir plusieurs nœuds enfant mais chaque nœud enfant ne peut avoir qu'un seul nœud parent.

Tous les attributs d'un enregistrement de données spécifique sont répertoriés sous un type d'entité. Chaque enregistrement de données est représenté sous forme d'une ligne et chaque attribut sous forme d'une colonne. Par exemple, le registre Windows est une base de données hiérarchique qui stocke les paramètres de configuration et options du système d'exploitation.

Ce modèle reconnu comme étant le premier modèle de base de données est créé par IBM dans les années 60. Plusieurs implémentations existent de nos jours.

3.2.3.1 L'agrégation d'arborescence

Les arborescences et même les graphes aléatoires (après dé-normalisation) peuvent être modélisés en tant qu'enregistrement de données unique ou document unique :

1- Cette technique est efficace si on accède à l'arborescence en une fois (dans la Figure 35, on constate une arborescence entière de commentaires d'un blog, permettant d'afficher une page spécifique d'un article).
1- La recherche et l'accès aléatoire aux données d'entrée sont considérés comme problématiques.
2- Les mises-à-jour sont inefficaces dans la plupart des implémentations NoSQL (comparé aux nœuds indépendants).

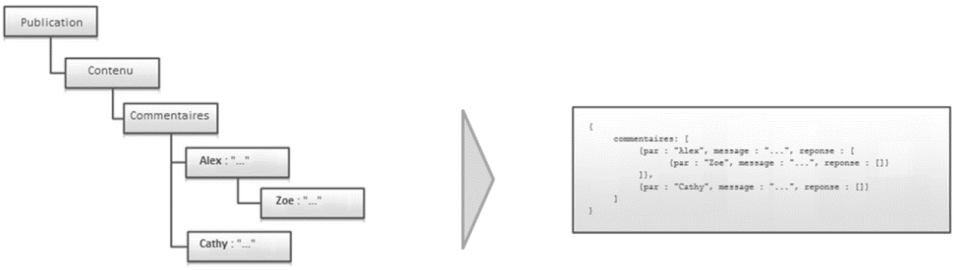

Figure 35 : Agrégation d'arborescence

La technique d'agrégation d'arborescence s'applique dans les stockages clé-valeur et les bases de données orientées document.

3.2.3.2 Les listes d'adjacence

Les listes d'adjacence sont une technique simple de modélisation de graphes. Chaque nœud est modélisé en tant qu'enregistrement de données indépendant contenant les tableaux d'ascendants ou de descendants directs. Cela permet de rechercher les nœuds par les identifiants de leurs parents ou enfants et de parcourir le graphe en faisant un saut par requête. Cette approche est habituellement inefficace, sauf si l'on possède la branche d'arborescence complète d'un nœud donné à disposition pour des traversées en largeur ou en profondeur.

Cette technique s'applique dans les stockages clé-valeur et les bases de données orientées document.

3.2.3.3 Les chemins énumérés

Cette technique consiste à stocker la chaîne des parents dans chaque nœud. Cela aide à éviter les traversées récursives des structures en arborescence. Cette technique peut être considérée comme une dérivée de la dé-normalisation dont l'idée est d'attribuer à chaque nœud la liste des identifiants de ses ascendants ou descendants afin de déterminer rapidement et sans traversée les parents et enfants correspondant au nœud.

Cette approche est particulièrement utile pour les moteurs de recherche plein texte, parce qu'elle permet de convertir les structures hiérarchiques en document texte plat. La Figure 36, permet de constater que tous les produits ou sous-catégories dans la catégorie Chaussures Homme peuvent être récupérés en utilisant une requête courte qui est simplement le nom de la catégorie.

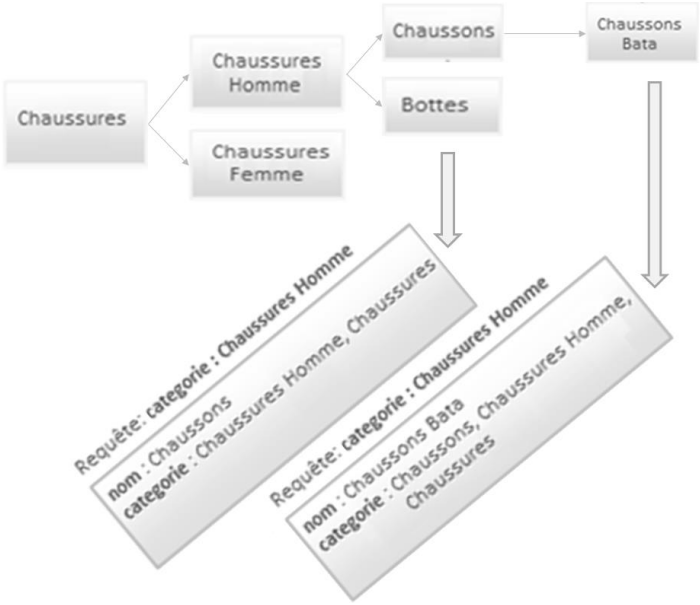

Figure 36 : Chemins énumérés de la hiérarchie des catégories d'un site marchand

Les chemins énumérés peuvent être stockés en tant que liste d'identifiants ou en tant que chaîne de caractères unique d'identifiants concaténés ensemble. Il est possible également de rechercher les nœuds correspondant à un critère déterminé de chemins partiels en utilisant des expressions régulières. La Figure 37 illustre cette situation.

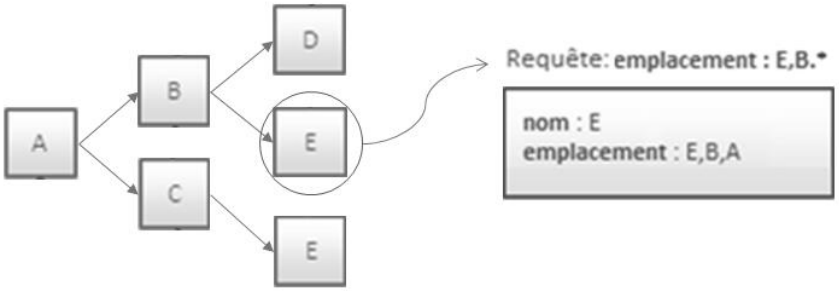

Figure 37 : Utilisation des expressions régulières pour parcourir les chemins énumérés

Finalement, la technique des chemins énumérés s'applique dans les stockages clé-valeur, les bases de données orientées document et les moteurs de recherche.

3.2.3.4 Les ensembles imbriqués

Cette approche fait partie également des techniques de modélisation hiérarchique. Elle consiste à modéliser les structures en arborescence et est utilisée dans les SGBD relationnels. Ce modèle est parfaitement applicable dans les modèles de stockage clé-valeur et les modèles de données orientés document. Le mécanisme de cette technique consiste à stocker les éléments d'une branche d'une arborescence dans un tableau, puis de cartographier chaque nœud sans branches dans un ensemble d'éléments en utilisant les index de début et de fin comme illustré dans la Figure 38.

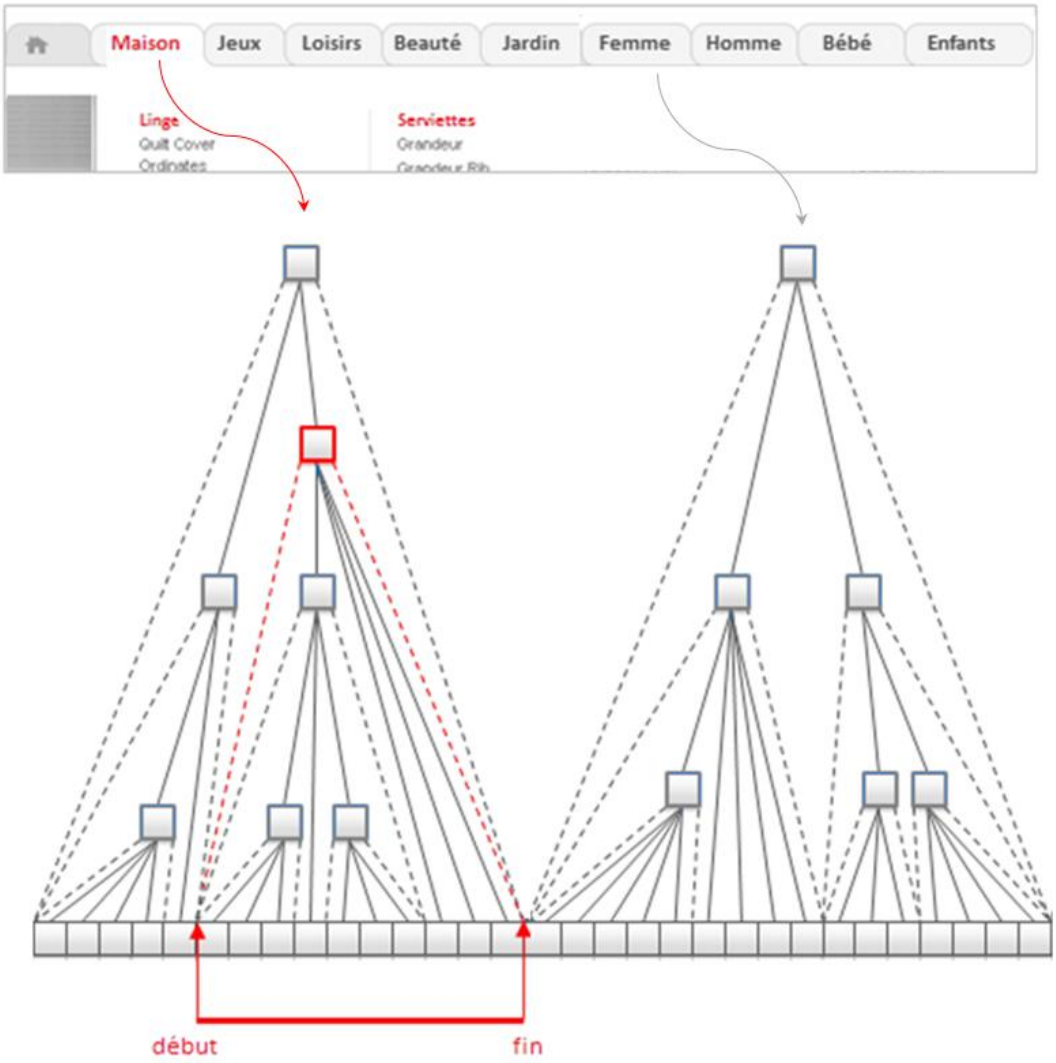

Figure 38 : Modélisation d'un catalogue de site marchand en utilisant les ensembles imbriqués

Finalement, cette technique s'applique dans les stockages clé-valeur et les bases de données orientées document.

3.2.3.5 L'aplatissement des documents imbriqués

Cette technique a pour objectif d'attribuer des documents texte plat aux entités métier dans le cadre du fonctionnement des moteurs de recherche utilisant ce type de document. Elle est appliquée selon une numérotation des noms des champs. Cela peut être relativement complexe si les entités disposent de structures internes complexes. Dans l'exemple de la Figure 39, chaque entité métier contient des informations identitaires (le nom de la personne et une liste de ses niveaux de compétences). Une façon simple de modéliser chaque entité, serait de créer un document texte plat avec les champs « competence » et « niveau ». Ce modèle permet de rechercher une personne par compétence ou par niveau mais les requêtes permettant de combiner les deux champs sont susceptibles de comporter de faux résultats de correspondance.

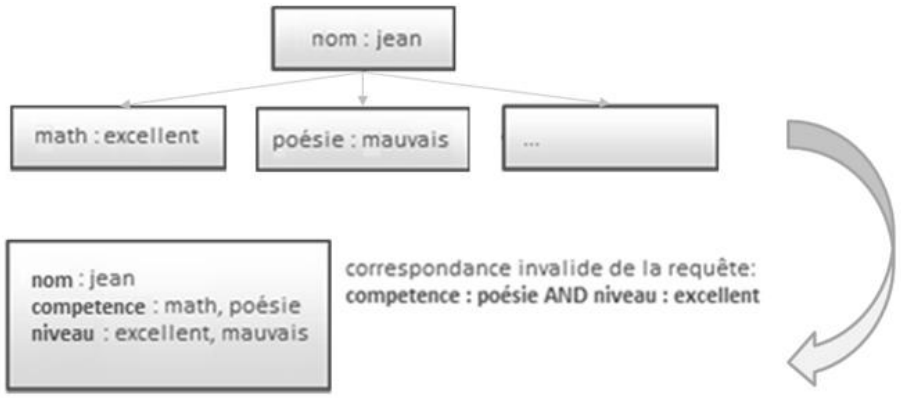

Figure 39 : Problème des documents imbriqués

Afin de contourner ce problème il faut indexer chaque élément « competence » et « niveau » correspondant, en tant qu'un couple dédié de champs « competence_i » et « niveau_i » et de pratiquer la recherche par la suite, pour tous ces couples simultanément (où le nombre d'opérations de type ou dans la requête est aussi élevé que le nombre d'éléments niveau d'une personne). Cette approche n'est pas vraiment évolutive parce que la complexité de la requête augmente rapidement et est relative au nombre de structures imbriqués. Cette approche est illustrée dans la Figure 40.

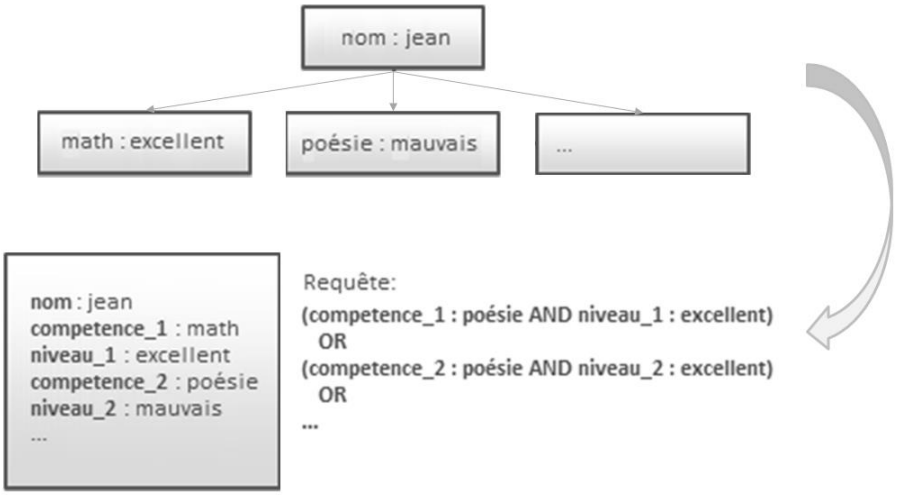

Figure 40 : Modélisation des documents imbriqués par numérotation des noms de champs

Par ailleurs, il est possible de procéder à la modélisation des documents imbriqués par requêtes de proximité. Cette implémentation apporte une solution au problème des documents imbriqués dont l'idée consiste à utiliser des requêtes de proximité qui limitent la distance acceptable entre les mots dans un document. Dans l'exemple de la Figure 41, tous les éléments « competence » et « niveau » sont indexés dans un seul champ nommé « competence_niveau ».

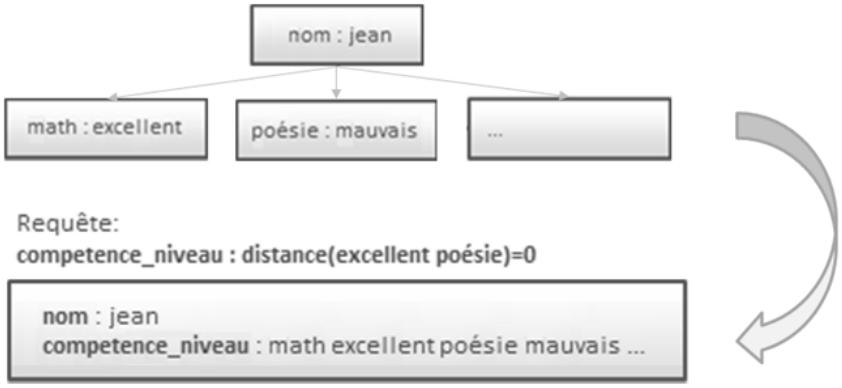

Figure 41 : Modélisation des documents imbriqués par requêtes de proximité

Ces techniques s'appliquent dans les moteurs de recherche uniquement.

3.2.3.6 Le traitement des graphes

Les bases de données orientées graphe sont très performantes pour le parcours de voisinage d'un nœud donné ou le parcours des liaisons entre deux ou plusieurs nœuds. Néanmoins, le traitement global des bases de données orientées graphe volumineuses est handicapé par le manque d'évolutivité de ces bases de données.

La technique de traitement des graphes en mode Batch relative aux bases de données orientées graphe peut être réalisée en utilisant les routines MapReduce [42] dans le but d'exploiter le voisinage d'un nœud donné ou la liaison entre deux ou plusieurs nœuds. Cette approche rend les modèles de données de type stockage clé-valeur, BigTable ou orienté document capables de traiter de larges données de graphes.

La représentation sous forme de listes d'adjacence peut être utilisée dans le traitement des graphes. Les modèles de données graphes, subissent une sérialisation en couples clé-valeur, en utilisant l'identifiant du sommet comme clé et l'enregistrement de données comprenant sa structure comme valeur. Dans le processus MapReduce, les phases Shuffle et Sort peuvent être exploitées pour propager l'information entre les sommets, en utilisant une forme de passation de message distribué. Dans la phase Reduce, tous les messages ayant la même clé, arrivent ensemble. Un nouveau comptage est alors réalisé.

En MapReduce, les éléments « Combiners » sont à l'origine des agrégations locales qui réduisent la quantité des données à distribuer, à travers le réseau. Ils prennent effet si et seulement si, il existe plusieurs couples clé-valeur à agréger avec la même clé calculée sur la même machine.

Cette technique illustrée dans la Figure 42 s'applique dans les stockages clé-valeur, les bases de données BigTable et les bases de données orientées document.

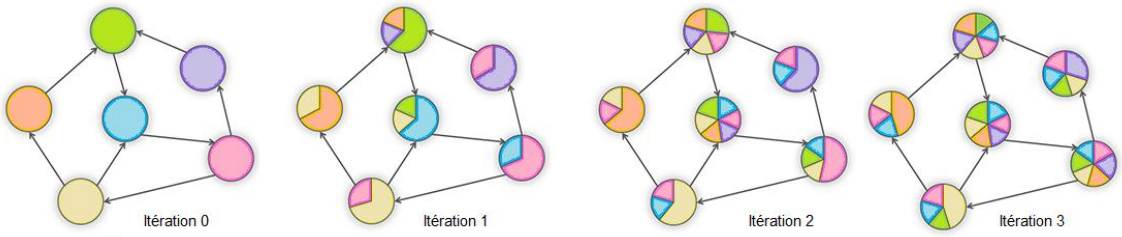

Figure 42 : Technique de traitement des graphes avec MapReduce

3.2.4 Conclusion

Le Tableau 2 récapitule l'usage des différentes techniques de modélisation citées précédemment, selon le type de la base de données non-relationnelle.

		Stockage clé-valeur	Base orientée colonne	Base orientée document	Base orientée graphe	Moteur de recherche
Techniques conceptuelles	Dé-normalisation	x	x	x		
	Agrégations	x	x	x		
	Jointures côté application	x	x	x	x	
Techniques générales	Agrégations atomiques	x	x	x		
	Clés énumérables	x				
	Réduction dimensionnelle	x	x	x		
	Tables d'index		x			
	Clé d'index composite		x			
	Agrégation avec des clés composites	x	x			
	Recherche inversée et agrégation directe	x	x	x		
Techniques hiérarchiques	Agrégation d'arborescence	x		x		
	Listes d'adjacence	x		x		
	Chemins énumérés	x		x		x
	Ensembles imbriqués	x		x		
	Aplatissement des documents imbriqués					x
	Traitement des graphes	x	x	x		

Tableau 2 : Récapitulatif des techniques de modélisation

3.3 L'approche de la modélisation intégratrice

Le défi le plus important de nos jours est d'obtenir des résultats de traitement de bonne qualité, avec un temps de traitement raisonnable et des coûts réduits. Pour ce faire, la chaîne de traitement doit être modifiée en amont dans le but d'y intégrer un ou plusieurs opérateurs de modélisation. Malheureusement, les modèles de traitement existant aujourd'hui ne prennent pas en considération cet aspect et se focalisent sur l'optimisation du temps de traitement des données en aval afin de réduire la durée et les coûts.

3.3.1 La modification de la chaîne de traitement

Cette étude vise à fournir une technique de pré-traitement pouvant donner une indication qui tend vers un résultat final précis qu'on ne peut obtenir sans traitement complet. Elle est basée sur une sélection d'opérateurs de modélisation qui une fois groupés ensembles selon un certain paramétrage, permettent de lancer un pré-traitement en amont puis d'obtenir cette indication suite à un traitement de volumétrie réduite des données.

Les outils et opérateurs de modélisation requis aideront l'utilisateur / développeur à identifier les champs de traitement en début de chaîne et d'envoyer dans le moteur de calcul seulement les données nécessaires pour obtenir le résultat demandé. Les autres données ne sont pas indispensables et doivent être isolées du traitement. Ainsi, la quantité des données traitées sera inférieure et le calcul sera moins gourmand en termes de temps et de ressources. Comme illustré dans la Figure 43, la netteté du résultat final est moins importante dans le cas du pré-traitement, que celui du traitement standard BigData. Cela s'explique par la nature des données contribuant à obtenir la valeur finale du calcul. Mais cela n'est pas gênant du moment qu'on retrouve un équilibre suffisamment utile pour le cas d'emploi, entre la netteté de l'indication et la durée de traitement qui lui est inversement proportionnelle.

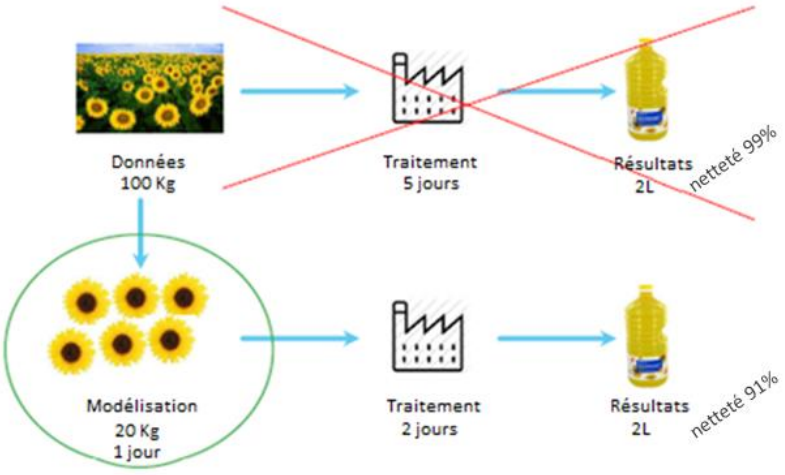

Figure 43 : Concept de pré-traitement

La seconde contribution consiste à réduire le périmètre de travail des industriels de calcul des données sur le Cloud au niveau de leurs politiques de prix, en décentralisant le traitement sur un ou plusieurs moteurs de traitement, en parallèle ou séquentiellement, tout en se basant sur le meilleur prix trouvé à l'instant « t ».

3.3.2 Conclusion

Par ailleurs, dans une approche de pré-traitement, on définit une chaîne de calcul en trois étapes :

1- L'acquisition des données diverses et variées, de sources multiples, de tailles et de formats différents.
2- Le formatage des données via des opérateurs de modélisation sélectionnés par l'utilisateur selon une configuration précise en fonction du calcul.
3- Le traitement des données présélectionnées par les opérateurs de modélisation dans le moteur Hadoop, via le processus classique de MapReduce.

La Figure 44 illustre ces trois étapes.

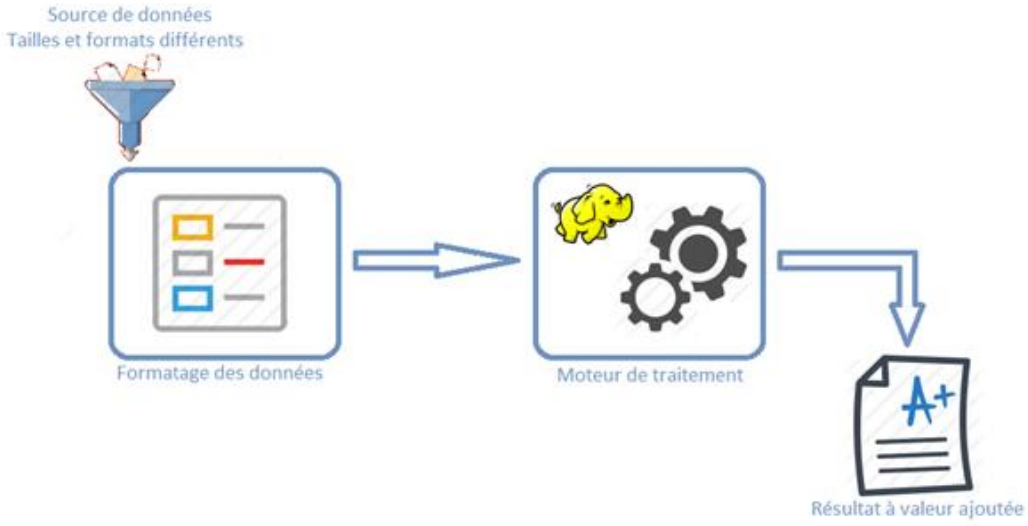

Figure 44 : Chaîne de pré-traitement

3.4 Conclusion du chapitre

Le modèle de données fournit une méthode de gestion visuelle des données en entrée et permet de créer une architecture de données fondamentale afin que l'on puisse avoir une multitude d'applications, permettant une meilleure réutilisation des données et une réduction des coûts. Chaque technique dispose de points forts et moins forts dans sa façon de s'adresser à chaque usager. La plupart est davantage orientée vers les utilisateurs avancés, plutôt que la communauté élargie non-spécialiste. Ces techniques produisent des modèles très complexes et se donnent comme objectif de couvrir toutes les contraintes d'utilisation. Par conséquent, cela arrive au détriment de la simplicité de la solution.

La facilité d'utilisation d'une technique est potentialisée par un traitement graphique des entités et des liaisons, ainsi que son adhésion aux principes de convivialité et d'interface utilisateur. Afin de faire une comparaison, l'évaluation doit considérer à la fois l'aspect complet de chaque technique et sa facilité d'implémentation. Les outils doivent permettre de manipuler :

1- Les entités et les attributs.
2- Les liaisons.
3- Les identifiants uniques.
4- Les sous-types et les super-types.
5- Les contraintes entre les liaisons.

La complexité d'une base de données relationnelle est limitée par l'évolutivité des stockages de données. En revanche, elle est caractérisée par la facilité d'interroger ses données. Les bases de données non-relationnelles sont complètement à l'opposé, avec une évolutivité illimitée et des capacités d'interrogation réduites.

Le défi du BigData est la facilité d'interrogation des données. L'application de la modélisation sur les données physiques facilite la gestion des enregistrements de données. L'avenir apportera des systèmes plus hybrides adaptés aux deux approches. Cependant, le modèle dynamique discuté reste un premier pas dans cette direction.

Chapitre 4. Les algorithmes de modélisation avec Hadoop MapReduce

4.1 Introduction au chapitre

MapReduce permet de manipuler de grandes quantités de données distribuées dans un Cluster de serveurs pour traitement. Ce modèle connaît un succès important auprès de sociétés possédant des centres de traitement des données et commence aussi à être utilisé au sein du traitement dans le Cloud. De nombreuses implémentations ont vu le jour dont la plus connue est Hadoop, programmé par Apache Software Foundation.

Dans le chapitre 3, on a vu les différentes techniques et les patrons de modélisation, regroupés dans trois grandes familles, les techniques conceptuelles, les techniques générales et les techniques hiérarchiques. Dans les paragraphes qui suivent, on va expliquer plusieurs patrons et algorithmes MapReduce correspondants afin de fournir une vue systématique des différentes techniques disponibles. Plusieurs cas d'emploi sont également exposés, pour lesquels on fournit les descriptions et les codes sources du modèle standard de Hadoop MapReduce. Enfin, on évoquera les potentiels de MapReduce dans le domaine de l'intelligence artificielle et en particulier l'apprentissage automatique.

4.1.1 L'algorithme MapReduce (rappel)

MapReduce est un modèle de programmation popularisé par Google (ayant obtenu un brevet sur la fonction MapReduce, mais dont la validité a été contestée). Il est principalement utilisé pour la manipulation et le traitement d'un nombre important de données au sein d'un Cluster. Le modèle standard de Hadoop MapReduce est composé de deux fonctions principales « map » et « reduce ». Il utilise une architecture de type maître-esclave, où un nœud maître dirige tous les nœuds esclaves.

L'algorithme MapReduce représenté dans la Figure 45 implémente les fonctionnalités de parallélisme automatique des programmes Hadoop, de gestion transparente du mode distribué et de résilience. Son programme se divise en général en plusieurs parties :

1- Les « Mappers » pour la lecture et le traitement.
2- Les « Reducers » pour la consolidation.
3- Les « Combiners » pour la semi-consolidation.
4- Les « Partitioners » pour la division.
5- Le « Sorting » pour le tri.

Dans les paragraphes suivant, sont exposées les différentes techniques d'implémentation de MapReduce [43].

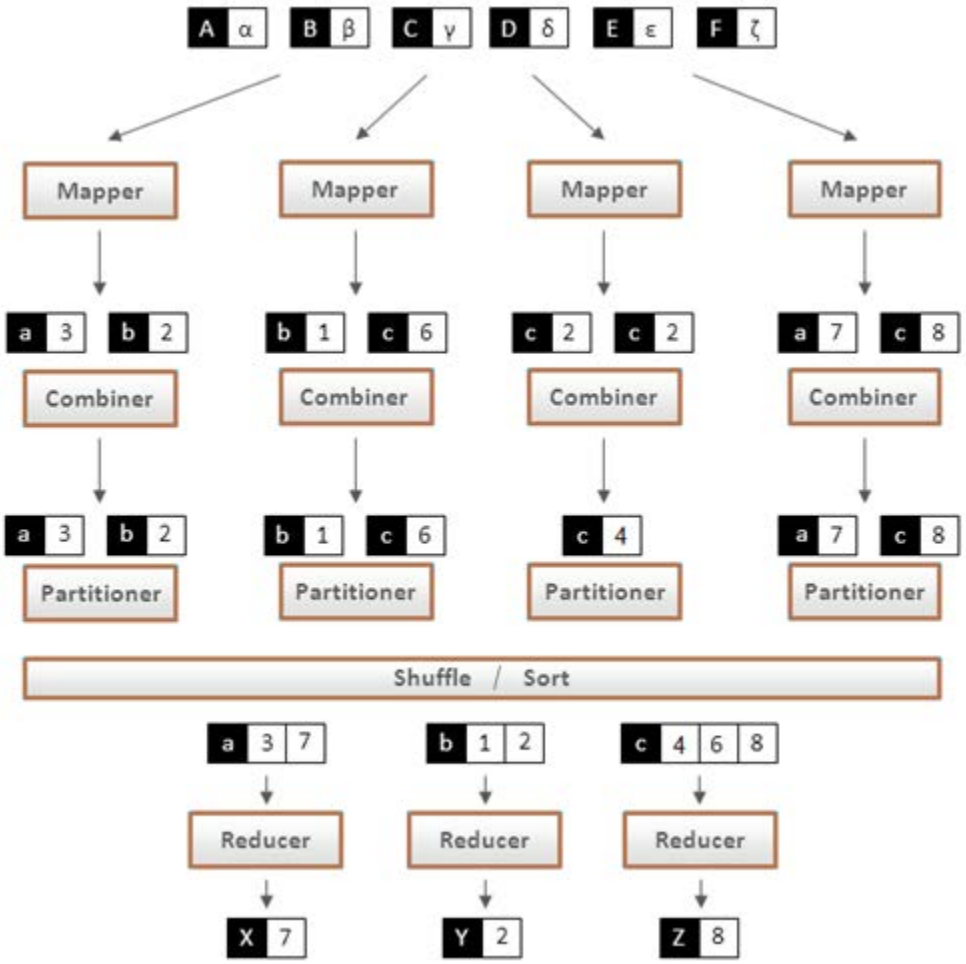

Figure 45 : Framework MapReduce

4.2 Les algorithmes correspondant aux principaux opérateurs de modélisation

Étant donné « E » l'ensemble des données, qu'elles proviennent de la source de données ou qu'elles soient des résultats de calcul, on définit la fonction « S » de « E » dans « E » pour chacun des opérateurs cités par la suite.

4.2.1 La transformation

Afin d'implémenter l'opérateur de transformation, dit Mapping, on définit :

$$S(d), \forall (d1 \dots di) \in E, S(d) = r \in E$$

```
public class TextCSVtoArrayList {
    public static void main(String[] args) {
        BufferedReader textBuffer = null;
        try {
            String textLine;
            textBuffer = new BufferedReader(new FileReader("/CSV-to-ArrayList.txt"));
            while ((textLine = textBuffer.readLine()) != null) {
                System.out.println("CSV data: " + textLine);
                System.out.println("ArrayList data: " + textCSVtoArrayList(textLine) + "\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (textBuffer != null) textBuffer.close();
            } catch (IOException textException) {
                textException.printStackTrace();
            }
        }
    }

    public static ArrayList < String > textCSVtoArrayList(String textCSV) {
        ArrayList < String > textResult = new ArrayList < String > ();
        if (textCSV != null) {
            String[] splitData = textCSV.split("\\s*,\\s*");
            for (int i = 0; i < splitData.length; i++) {
                if (!(splitData[i] == null) || !(splitData[i].length() == 0)) {
                    textResult.add(splitData[i].trim());
                }
            }
        }
        return textResult;
    }
}
```

4.2.2 Le filtre

Afin d'implémenter l'opérateur de filtre, dit Filter, on définit :

$$S(d), \forall (d1 \dots di) \in E, S(\{d\}) = r \in E$$

```
public class FileFilter extends Configured implements PathFilter {

    Configuration conf;
    FileSystem fs;

    @Override
    public boolean accept(Path path) {
        try {
            if (fs.isDirectory(path)) { return true; }
        } catch (IOException e1) {
            e1.printStackTrace();
            return false;
        }
        if (conf.get("file.mtime") != null) {
            int mTime = 0;
            String strMtime = conf.get("file.mtime");
            mTime = Integer.valueOf(strMtime.substring(1, strMtime.length()));
            try {
                FileStatus file = fs.getFileStatus(path);
                long now = System.currentTimeMillis() / (1000 * 3600 * 24);
                long time = file.getModificationTime() / (1000 * 3600 * 24);
                long lastModifTime = now - time;
                boolean accept;
                if (strMtime.charAt(0) == '-') {
                    accept = mTime < lastModifTime ? true : false;
                    System.out.println("File " + path.toString() +
                                       " modified " + lastModifTime +
                                       " days ago, is " + mTime + " lower ? " + accept);
                } else {
                    accept = mTime > lastModifTime ? true : false;
                    System.out.println("File " + path.toString() +
                                       " modified " + lastModifTime +
                                       " days ago, is " + mTime + " greater ? " + accept);
                }
                return accept;
            } catch (IOException e) {
                e.printStackTrace();
                return false;
            }
        } else { return true; }
    }

    @Override
    public void setConf(Configuration conf) {
        this.conf = conf;
        if (conf != null) {
            try { fs = FileSystem.get(conf); }
            catch (IOException e) { e.printStackTrace(); }
        }
    }
}
```

4.2.3 Le découpage

Afin d'implémenter l'opérateur de découpage, dit Split, on définit :

$$S(d), \forall (d1 \dots di) \in E, S(d) = \{r\} \in E$$

```
public class SplitFile {
    int FRG_FSIZE = 0;

    public File[] splitFile(File source, int noFile, int lsize) {
        FRG_FSIZE = 1024 * 5;
        File[] fileFragments = new File[noFile];
        String[] frgfName = new String[noFile];
        try {
            String sourceFName = source.getName();
            long sourceFSize = source.length();
            FileInputStream fis = new FileInputStream(source);
            String Fileinfo = new String(sourceFName + "," + String.valueOf(sourceFSize));
            System.out.println(noFile);
            if (lsize != 0) {
                noFile--;
            }
            System.out.println(noFile);
            sourceFName = sourceFName.substring(0, sourceFName.lastIndexOf("."));
            int j = 0;
            for (int i = 1; i <= noFile; i++) {
                frgfName[i - 1] = "temp\\" + sourceFName + String.valueOf(i) + ".splt";
                fileFragments[i - 1] = new File(frgfName[i - 1]);
                FileOutputStream fos = new FileOutputStream(fileFragments[i - 1]);
                byte[] data = new byte[FRG_FSIZE];
                int count = fis.read(data);
                fos.write(data);
                fos.close();
                String frgFileInfo = new String(frgfName[i - 1] +
                                                "," + String.valueOf(FRG_FSIZE));
            }
            if (lsize != 0) {
                System.out.println(noFile);
                frgfName[noFile] = "temp\\" + sourceFName +
                                   String.valueOf(noFile + 1) + ".splt";
                fileFragments[noFile] = new File(frgfName[noFile]);
                FileOutputStream fos = new FileOutputStream(fileFragments[noFile]);
                byte[] data = new byte[lsize];
                int count = fis.read(data);
                fos.write(data);
                fos.close();
                String frgFileInfo = new String(frgfName[noFile] +
                                                "," + String.valueOf(lsize));
            }
        } catch (Exception e) {
            System.out.println("Error in Splitting" + e);
            JOptionPane.showMessageDialog(null, e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            return null;
        }
        return fileFragments;
    }

    public static void main(String ar[]) {
        SplitFile sf = new SplitFile();
        sf.splitFile(new File("data.doc"), 5, 1024);
    }
}
```

4.2.4 La fusion

Afin d'implémenter l'opérateur de fusion, dit Merge, on définit :

$$S(d), \forall (d1 \dots di) \in E, S(\{d\}) = r \in E$$

```
public class MergeFile {
    public File mergeFiles() {
        try {
            File[] files = new File[5];
            for (int i = 1; i <= 5; i++) {
                String fname = "Data.doc";
                files[i - 1] = new File(fname);
            }
            File outFile = new File("Temp.doc");
            FileOutputStream fileOS = new FileOutputStream(outFile);
            for (int i = 0; i < files.length; i++) {
                FileInputStream fileIS = new FileInputStream(files[i]);
                byte[] data = new byte[(int) files[i].length()];
                int count = fileIS.read(data);
                fileOS.write(data);
                fileIS.close();
            }
            fileOS.close();
            return outFile;
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage(), "", JOptionPane.ERROR_MESSAGE);
        }
        return null;
    }

    public static void main(String ar[]) {
        MergeFile mf = new MergeFile();
        mf.mergeFiles();
    }
}
```

4.2.5 Conclusion

Dans ce paragraphe on a montré des exemples d'implémentation des algorithmes correspondant aux principaux opérateurs de modélisation :

1- Le filtre, permettant d'isoler des données utiles pour un traitement particulier.
2- Le découpage, permettant de diviser un fichier volumineux en plusieurs morceaux.
3- La transformation, permettant de changer une structure de fichier de données en une autre.
4- La fusion, permettant de regrouper un ensemble de fichiers de moindre taille en un seul.

4.3 Les patrons basiques MapReduce

Ce paragraphe liste les implémentations basiques des patrons MapReduce exposés sous la forme de problématiques et solutions, ainsi que le code correspondant.

4.3.1 Le comptage et l'addition

Étant donné plusieurs documents dont chacun est constitué d'un ensemble de mots-clés, on va calculer le nombre total d'instances de chaque mot-clé dans tous les documents, sachant que chaque document est un fichier journal où chaque enregistrement contient un temps de réponse dont la moyenne est à calculer également.

Le code suivant est composé simplement d'un « Mapper » qui renvoie un compteur à chaque traitement de mot-clé, ainsi qu'un « Reducer » qui en fera la somme totale.

```
class Mapper
    method Map(docid id, doc d)
        for all term t in doc d do
            Emit(term t, count 1)

class Reducer
    method Reduce(term t, counts[c1, c2, ...])
        sum = 0
        for all count c in [c1, c2, ...] do
            sum = sum + c
        Emit(term t, count sum)
```

L'inconvénient de cette approche est le nombre élevé de faux compteurs dans le « Mapper » susceptibles de décrémenter ce nombre avec une addition dans chaque document.

```
class Mapper
    method Map(docid id, doc d)
        H = new AssociativeArray
        for all term t in doc d do
            H{t} = H {t} + 1
        for all term t in H do
            Emit(term t, count H{t})
```

Afin de regrouper les compteurs pour tous les documents en cours de traitement par une instance « Mapper », il est possible d'utiliser les « Combiners ».

```
class Mapper
    method Map(docid id, doc d)
        for all term t in doc d do Emit(term t, count 1)

class Combiner
    method Combine(term t, [c1, c2, ...])
        sum = 0
        for all count c in [c1, c2, ...] do sum = sum + c
        Emit(term t, count sum)

class Reducer
    method Reduce(term t, counts[c1, c2, ...])
        sum = 0
        for all count c in [c1, c2, ...] do sum = sum + c
        Emit(term t, count sum)
```

4.3.1.1 Le mode d'application

Ce scénario s'applique dans l'analyse de journal et l'interrogation de données.

4.3.2 L'assemblage

Étant donné un ensemble d'éléments dont chacun dispose d'une fonction spécifique, il faudrait sauvegarder tous les éléments ayant la même valeur de fonction dans un même fichier ou effectuer d'autres calculs nécessitant le traitement de ces éléments, en tant que groupe. L'exemple typique serait la création d'index inversés.

4.3.2.1 La solution

La solution est très simple. Le « Mapper » calcule la valeur de la fonction donnée pour chaque élément et renvoie cette valeur en tant que clé ainsi que l'élément correspondant en tant que valeur. Le « Reducer » regroupe alors tous les éléments par valeur et procède au traitement ou à la sauvegarde dans le fichier. Dans le cas des index inversés, les éléments seraient des mots-clés et la fonction serait l'identifiant du document où se trouve chaque mot-clé.

4.3.2.2 Le mode d'application

Ce scénario s'applique dans la création d'index inversés et la solution ETL.

4.3.3 Les filtres, l'analyse et la validation

Étant donné un ensemble d'enregistrements, il faudrait regrouper tous ceux parmi eux qui répondent à certaines conditions ou convertir chaque enregistrement, indépendamment des autres, à une autre représentation. Ce cas inclut des tâches d'analyse de texte, d'extraction de valeurs et de conversion d'un format à un autre.

4.3.3.1 La solution

Une solution simple consiste à ce que le « Mapper » traite les enregistrements un par un, en renvoyant les éléments correspondant aux conditions ou leurs instances converties.

4.3.3.2 Le mode d'application

Ce scénario s'applique dans l'analyse de journal, l'interrogation de données, la solution ETL et la validation des données.

4.3.4 L'exécution des tâches distribuées

Étant donné une importante problématique de calcul pouvant être divisée en plusieurs parties, il faudrait combiner les résultats correspondant ensemble afin d'obtenir un résultat final.

4.3.4.1 La solution

La description du problème correspond à un ensemble de spécifications stockées en données d'entrée des « Mappers ». Chaque « Mapper » traite les spécifications en faisant les calculs correspondants et en renvoyant le résultat. Le « Reducer » combine alors les différents résultats en un seul.

4.3.4.2 La simulation d'un système de communication numérique

Dans un système de communication numérique, on définit un logiciel simulateur qui traite un certain volume de données aléatoires à travers le modèle défini du système en calculant la probabilité d'erreur de débit. Chaque « Mapper » lance alors la simulation pour une quantité spécifique de données correspondant à « 1/n$^{ème"}$» de l'échantillonnage requis, en retournant le taux d'erreur. Le « Reducer » calcule alors la moyenne du taux d'erreur émis.

4.3.4.3 Le mode d'application

Ce scénario s'applique dans la simulation physique, la simulation d'ingénierie, l'analyse numérique et le test de performance.

4.3.5 Le tri

Soit un ensemble d'enregistrements, pour lesquels il faudrait procéder à un tri selon certaines règles ou un traitement selon un ordre spécifique.

4.3.5.1 La solution

On va établir une solution simple basée sur des « Mappers » qui renvoient tous les éléments en tant que valeurs associées avec les clés de tri assemblées, en tant que fonctions d'éléments. Cependant, le tri est souvent utilisé différemment, raison pour laquelle il est considéré comme le cœur du MapReduce. En particulier, il est possible d'utiliser les clés composites pour accomplir un tri secondaire et un groupement.

4.3.5.2 Le mode d'application

Dans MapReduce, le tri est prévu initialement pour trier des couples clé-valeur par clé. Il existe en revanche des techniques permettant de faire une mise à niveau de l'implémentation Hadoop MapReduce afin de procéder à un tri par valeur. Il est important de noter également que MapReduce est utilisé pour le tri des données d'origine et non pas les données intermédiaires. Dans les pratiques BigData, il est recommandé de maintenir les données triées tout au long de la chaîne de traitement. En d'autres termes, il est plus efficace et plus performant de procéder au tri des données à l'entrée, plutôt qu'à chaque requête MapReduce.

Ce scénario s'applique dans la solution ETL et l'analyse de données.

4.3.6 Conclusion

Dans ce paragraphe, on a décrit les patrons basiques MapReduce, tels que le comptage, l'addition, l'assemblage, le filtre et le tri, tout en exposant une problématique et la solution correspondant à chaque fois, ainsi que le mode d'application de cette solution.

4.4 Les patrons non-basiques MapReduce

Cette section décrit les implémentations non-basiques des patrons MapReduce exposés sous la forme de problématiques et solutions, ainsi que le code correspondant.

4.4.1 Le traitement des graphes

Étant donné un réseau d'entités liées entre elles, il faudrait calculer l'état de chaque entité en se basant sur les propriétés des entités voisines. Cet état peut représenter la distance entre les nœuds, une indication sur l'existence d'une entité voisine, les caractéristiques de densité de voisinage ou autres.

4.4.1.1 La solution

Un réseau est stocké sous la forme d'un ensemble de nœuds dont chacun contient une liste d'identifiants des nœuds adjacents. Conceptuellement, les programmes MapReduce sont performants en algorithmes itératifs. A chaque itération, chacun des nœuds envoie des messages à son voisinage leur permettant de mettre à jour leur statut en se basant sur le message reçu. Certaines conditions, telles qu'un non-changement d'état initial entre deux itérations ou une valeur maximale fixe définissant le diamètre du réseau, mettent fin aux itérations. D'un point de vue technique, le « Mapper » renvoie les messages pour chaque nœud en utilisant l'identifiant du nœud voisin comme clé. Par conséquent, tous les messages sont regroupés, permettant au « Reducer » de recalculer l'état et de procéder à une mise-à-jour du nœud.

```
class Mapper
    method Map(id n, object N)
        Emit(id n, object N)
        for all id m in N.OutgoingRelations do
            Emit(id m, message getMessage(N))

class Reducer
    method Reduce(id m, [s1, s2, ...])
        M = null
        messages = []
        for all s in [s1, s2, ...] do
            if IsObject(s) then M = s
            else messages.add(s)
            M.State = calculateState(messages)
            Emit(id m, item M)
```

Il est important de préciser que l'état d'un nœud se propage rapidement à travers le réseau, vu que chaque nœud recevant le message le retransmet directement à son voisinage, comme indiqué précédemment dans la Figure 42.

4.4.1.2 La propagation de la disponibilité à travers l'arborescence

Ce problème est inspiré de la vie quotidienne en e-commerce. Soit une arborescence qui s'étend, à travers les catégories générales (Homme, Femme, Enfant) jusqu'aux catégories particulière (Chemise, Manteau, Robe) et éventuellement des catégories très spécifiques (Chemise Bleue). Les catégories du dernier niveau sont disponibles (contenant des produits) ou pas. Les catégories du haut niveau sont considérées disponibles, si au moins une des catégories du dernier niveau de la sous-arborescence correspondant, est disponible.

L'objectif est de calculer la disponibilité de toutes les catégories générales, si celle des catégories du dernier niveau est connue. Ce problème peut être résolu avec la logique du paragraphe précédent. On définit les 2 méthodes « getMessage » et « calculateState » comme suit :

```
class N
    State in {True = 2, False = 1, null = 0}

    method getMessage(object N)
        return N.State

    method calculateState(state s, data [d1, d2,...])
        return max([d1, d2,...])
```

4.4.1.3 L'étendue de la recherche

Étant donné un graphe quelconque, il faudrait calculer la distance (en nombre de nœuds) entre le nœud source et tous les autres nœuds dans le graphe.

```
class N
    State in {source = 0, node = INFINITY}

    method getMessage(N)
        return N.State + 1

    method calculateState(state s, data [d1, d2,...])
        min([d1, d2,...])
```

4.4.1.4 L'évaluation des pages avec l'algorithme PageRank

Cet algorithme a été introduit par Google pour calculer l'évaluation d'une page sur Internet, en se basant sur le nombre des autres pages ayant des hyperliens avec cette dernière. Cet algorithme est d'une complexité élevée mais dans le fond, ce n'est qu'une simple propagation de poids entre les nœuds (pages), dont chacune calcule le sien selon les autres poids du voisinage.

```
class N
    State is PageRank

    method getMessage(object N)
        return N.State / N.OutgoingRelations.size()

    method calculateState(state s, data [d1, d2,...])
        return (sum([d1, d2,...]))
```

4.4.1.5 L'agrégation des données côté Mapper

Il est important de préciser que le schéma précédent est très générique et ne profite pas du fait que l'état d'un nœud est une valeur numérique. Dans la majorité des cas pratiques l'agrégation des valeurs peut être réalisée du côté du « Mapper » en mettant à profit cette particularité. Cette optimisation apparaît dans le code suivant pour l'algorithme d'évaluation des pages.

```
class Mapper
    method Initialize
        H = new AssociativeArray

    method Map(id n, object N)
        p = N.PageRank / N.OutgoingRelations.size()
        Emit(id n, object N)
        for all id m in N.OutgoingRelations do
            H{m} = H{m} + p

    method Close
        for all id n in H do
            Emit(id n, value H{n})

class Reducer
    method Reduce(id m, [s1, s2,...])
        M = null
        p = 0
        for all s in [s1, s2,...] do
            if IsObject(s) then
                M = s
            else
                p = p + s
        M.PageRank = p
        Emit(id m, item M)
```

4.4.1.6 Le mode d'application

Ce scénario s'applique dans l'analyse des graphes et l'indexation sur Internet.

4.4.2 Les valeurs distinctes

Étant donné un ensemble d'enregistrements contenant les champs « A » et « B », il faudrait calculer le nombre total des valeurs uniques du champ « A », pour chacun des sous-ensembles ayant la même valeur dans « B » (groupées). Ce problème peut être généralisé et reformulé, en termes de recherche.

4.4.2.1 La reformulation

Et Étant donné un ensemble d'enregistrements dont chacun contient un champ « A » et un nombre aléatoire de libellés « B = {B1, B2, …} », il faudrait calculer le nombre total des valeurs uniques du champ « A » pour chaque sous-ensemble d'enregistrements pour chaque valeur de chaque libellé :

$Enregistrement\ 1 :\ A = 1, B = \{x, y\}$
$Enregistrement\ 2 :\ A = 2, B = \{x, t, u\}$
$Enregistrement\ 3 :\ A = 1, B = \{y\}$
$Enregistrement\ 4 :\ A = 3, B = \{x, y\}$

$Résultat :$
$x -> 3 \quad // A = 1, A = 2, A = 3$
$y -> 2 \quad // A = 1, A = 3$
$t -> 1 \quad // A = 2$
$u -> 1 \quad // A = 2$

4.4.2.2 La première solution

La première approche serait de résoudre le problème en deux temps. Tout d'abord, le « Mapper » renvoie des faux compteurs pour chaque couple « A » et « B ». Le « Reducer » calcule le nombre total d'instances pour chacun des couples. L'objectif principal de cette étape est de garantir l'unicité des valeurs « A ». Dans un deuxième temps, les couples seront groupées par « B » et le nombre total des éléments dans chaque groupe sera calculé.

La phase 1 :

```
class Mapper
    method Map(null, record [value a, categories [b1, b2,...]])
        for all category b in [b1, b2,...]
            Emit(record [b, a], count 1)

class Reducer
    method Reduce(record [b, a], counts [n1, n2, ...])
        Emit(record [b, a], null )
```

La phase 2 :

```
class Mapper
    method Map(record [a, b], null)
        Emit(value b, count 1)

class Reducer
    method Reduce(value b, counts [n1, n2,...])
        Emit(value b, sum([n1, n2,...]))
```

4.4.2.3 La deuxième solution

Cette solution nécessite un seul programme MapReduce mais elle n'est pas très évolutive et son usage est limité. L'algorithme est simple :

1- Un « Mapper » qui renvoie les valeurs et les catégories.
2- Un « Reducer » qui exclue les doublons de la liste des catégories de chaque valeur et incrémente le compteur de chaque catégorie.
3- Finalement faire une somme de tous les compteurs renvoyés par le « Reducer ».

Cette approche est applicable si le nombre d'enregistrements avec la même valeur a et le nombre total des catégories ne sont pas très élevés. Par exemple on utilise cette solution pour le traitement de registres de publications Internet et la classification des utilisateurs (le nombre total des utilisateurs est élevé mais le nombre d'événements pour un utilisateur est limité, ainsi que le nombre de catégories de classification). Les « Combiners » peuvent être également utilisés pour exclure les doublons des listes de catégories avant l'envoi des données au « Reducer ».

```
class Mapper
    method Map(null, record [value a, categories [b1, b2,...] )
        for all category b in [b1, b2,...]
            Emit(value a, category b)

class Reducer
    method Initialize
        H = new AssociativeArray : category -> count

    method Reduce(value a, categories [b1, b2,...])
        [b1', b2',..] = ExcludeDuplicates( [b1, b2,..] )
        for all category b in [b1', b2',...]
            F{b} = F{b} + 1

    method Close
        for all category b in F do
            Emit(category b, count F{b})
```

4.4.2.4 Le mode d'application

Ce scénario s'applique dans l'analyse de journal d'activité utilisateur et le comptage d'utilisateurs uniques.

4.4.3 La corrélation croisée

Étant donné un ensemble de tuples d'éléments, il faudrait calculer pour un couple d'éléments possible, le nombre de tuples ayant les mêmes éléments. Si le nombre total d'éléments est « N », alors la valeur « N x N » sera renvoyée.

Ce problème apparaît dans les analyses de texte (chaque élément est un mot et chaque tuple est une phrase), les analyses de marché (le client qui achète ceci a tendance à acheter cela également). Si « N x N » est relativement petit et que la matrice correspondant peut être gérée dans la mémoire d'une seule machine, alors l'implémentation sera facile.

4.4.3.1 L'approche par couple

La première approche consiste à renvoyer tous les couples avec de faux compteurs par le « Mapper », puis de faire la somme de ces compteurs dans le « Reducer » :

1- Les « Combiners » n'ont pas de valeur ajoutée, vu que tous les couples sont distincts en général.
2- Il n'y a pas d'accumulation en mémoire pouvant causer une saturation.

```
class Mapper
    method Map(null, items [i1, i2,...] )
        for all item i in [i1, i2,...]
            for all item j in [i1, i2,...]
                Emit(pair [i j], count 1)

class Reducer
    method Reduce(pair [i j], counts [c1, c2,...])
        s = sum([c1, c2,...])
        Emit(pair[i j], count s)
```

4.4.3.2 L'approche par ligne

Dans cette approche, on groupe les données par le premier élément en couple et on maintient un tableau associatif (Ligne), où les compteurs de tous les éléments en voisinage sont cumulés. Le « Reducer » reçoit toutes les Lignes de l'élément en cours, les fusionne et renvoie le même résultat que celui de l'approche par couple. On constate alors les choses suivantes :

1- Cela génère moins de clés intermédiaires, puisqu'il y a moins de tris à faire.
2- Cette approche bénéficie beaucoup des « Combiners ».
3- L'accumulation en mémoire peut causer des problèmes, si l'implémentation n'est pas de bonne qualité.
4- Elle nécessite une implémentation plus complexe.
5- D'une manière générale, cette approche est plus performante.

```
class Mapper
    method Map(null, items [i1, i2,...] )
        for all item i in [i1, i2,...]
            H = new AssociativeArray : item -> counter
            for all item j in [i1, i2,...]
                H{j} = H{j} + 1
            Emit(item i, stripe H)

class Reducer
    method Reduce(item i, stripes [H1, H2,...])
        H = new AssociativeArray : item -> counter
        H = merge-sum([H1, H2,...])
        for all item j in H.keys()
            Emit(pair [i j], H{j})
```

4.4.3.3 Le mode d'application

Ce scénario s'applique dans l'analyse de texte et l'analyse de marché.

4.4.4 Conclusion

Cette section a décrit les patrons non-basiques MapReduce et notamment le traitement des graphes, les valeurs distinctes et la corrélation croisée, tout en exposant une problématique et la solution correspondante à chaque fois, ainsi que le mode d'application de cette solution.

4.5 Les patrons relationnels MapReduce

Dans cette section on va parcourir les principaux opérateurs relationnels et discuter de leur implémentation dans les programmes MapReduce.

4.5.1 La sélection

Voici un exemple de code correspondant :

```
class Mapper
    method Map(rowkey key, tuple t)
        if t satisfies the predicate Emit(tuple t, null)
```

4.5.2 La projection

La projection est un petit peu plus complexe que la sélection. Il suffit d'utiliser un « Reducer » pour éliminer les doublons possibles.

```
class Mapper
    method Map(rowkey key, tuple t)
        tuple g = project(t)
        Emit(tuple g, null)

class Reducer
    method Reduce(tuple t, array n)
        Emit(tuple t, null)
```

4.5.3 L'union

Les « Mappers » sont alors alimentés par tous les enregistrements de deux ensembles à unifier. Le « Reducer » est utilisé pour éliminer les doublons.

```
class Mapper
    method Map(rowkey key, tuple t)
        Emit(tuple t, null)

class Reducer
    method Reduce(tuple t, array n)
        Emit(tuple t, null)
```

4.5.4 L'intersection

Les « Mappers » sont alors alimentés par les enregistrements de deux ensembles à croiser. Le « Reducer » renvoie seulement les enregistrements qui apparaissent deux fois. Cela est possible seulement si les deux ensembles contiennent ces enregistrements composés de clés primaires, ne pouvant donc pas appartenir deux fois au même ensemble.

```
class Mapper
    method Map(rowkey key, tuple t)
        Emit(tuple t, null)

class Reducer
    method Reduce(tuple t, array n)
        if n.size() = 2
            Emit(tuple t, null)
```

4.5.5 La différence

Soit 2 ensembles d'enregistrements « A » et « B ». Pour calculer la différence entre « A » et « B », les « Mappers » renvoient tous les tuples et les étiquettent par le nom de l'ensemble auquel ils appartiennent. Le « Reducer » renvoie alors seulement les enregistrements appartenant à « A » mais pas à « B ».

```
class Mapper
    method Map(rowkey key, tuple t)
        Emit(tuple t, string t.SetName)

class Reducer
    method Reduce(tuple t, array n)
        if n.size() = 1 and n[1] = 'A'
            Emit(tuple t, null)
```

4.5.6 Le groupement et l'agrégation

Les groupements et les agrégations peuvent être réalisés avec un seul programme MapReduce. Le « Mapper » extrait de chaque tuple les valeurs à grouper ou à agréger et les limite. Le « Reducer » reçoit les valeurs à agréger déjà groupées et calcule la fonction d'agrégation. Les fonctions d'agrégation typiques comme « sum » et « max » peuvent être calculées façon diffusion, ce qui ne nécessite pas de gérer toutes les valeurs simultanément.

Cependant dans certains cas, un programme MapReduce en deux temps est nécessaire, par exemple le patron Distinct Value.

```
class Mapper
    method Map(null, tuple [value GroupBy, value AggregateBy, value ...])
        Emit(value GroupBy, value AggregateBy)

class Reducer
    method Reduce(value GroupBy, [v1, v2,...])
        Emit(value GroupBy, aggregate([v1, v2,...]))
```

4.5.7 Les jointures

Les jointures sont tout à fait possibles avec MapReduce. Il existe cependant plusieurs techniques à des degrés d'efficacité différents et relatives au volume de données à traiter [44] [45]. Dans ce paragraphe, on va parler de 2 types de jointures :

1- La jointure de répartition, dite Reduce Sort-Merge Join.
2- La jointure répliquée, dite Map Hash Join.

4.5.7.1 La jointure de répartition

Étant donné un algorithme de jointure qui s'applique entre 2 ensembles « A » et « B » sur une clé « K », le « Mapper » parcourt alors tous les tuples dans « A » et « B », extrait la clé « K » du tuple, l'étiquette avec le nom de l'ensemble auquel il appartient « A » ou « B », puis renvoie le tuple étiqueté, en utilisant « K » comme clé. Le « Reducer » reçoit tous les tuples d'une clé donnée « K » et les place dans 2 espaces-tampon pour « A » et pour « B ».

Quand ces deux espaces-tampon sont pleins, le « Reducer » lance une boucle imbriquée là-dessus et renvoie une jointure croisée des deux espaces-tampon. Chaque tuple renvoyé est une concaténation des valeurs « A-uplet », « B-uplet » et « K ». Cette approche présente en revanche les inconvénients suivants :

1- Le « Mapper » envoie toutes les données, même pour les clés instanciées dans un seul sous-ensemble et ne disposant pas d'un équivalent dans l'autre.
2- Le « Reducer » doit garder toutes les données relatives à une clé en mémoire et doit les gérer dans une mémoire Swap dans le cas où la mémoire initiale n'est pas suffisante.

Cependant, une jointure de répartition est une technique générique pouvant être utilisée sans problème quand les autres techniques optimisées ne sont pas applicables.

```
class Mapper
    method Map(null, tuple [join_key k, value v1, value v2,...])
        Emit(join_key k, tagged_tuple [set_name tag, values [v1, v2, ...]])

class Reducer
    method Reduce(join_key k, tagged_tuples [t1, t2,...])
        F = new AssociativeArray : set_name -> values
        for all tagged_tuple t in [t1, t2,...]
            F{t.tag}.add(t.values)
        for all values a in F{'A'}
            for all values b in F{'B'}
                Emit(null, [k a b])
```

4.5.7.2 La jointure répliquée

En pratique il est tout à fait possible de procéder à une jointure entre un ensemble de petite taille et un autre de taille plus importante (comme une liste d'utilisateurs et une liste de fichiers journal).

Soit une jointure entre 2 ensembles « A » et « B », dont « A » est relativement de petite taille. L'ensemble « A » peut être alors réparti sur les « Mappers », dont chacun charge un élément et l'index avec la clé de jointure. Dans ce cas la technique la plus efficace à utiliser est la table de hachage (Hash). Ensuite, le « Mapper » parcourt les tuples de l'ensemble « B » et applique la jointure avec les tuples correspondant dans « A », stockés dans la table de hachage. Cette approche est très efficace, vu qu'il n'y a pas besoin de procéder à un tri ou une diffusion de l'ensemble « B », à travers le réseau. La taille petite de l'ensemble « A » en revanche, lui permet de profiter d'une meilleure répartition sur les « Mappers ».

```
class Mapper
    method Initialize
        F = new AssociativeArray : join_key -> tuple from A
        A = loadA()
        for all [ join_key k, tuple [a1, a2,...]] in A
            F{k} = F{k}.append([a1, a2,...])

    method Map(join_key k, tuple b)
        for all tuple a in F{k}
            Emit(null, tuple [k a b])
```

4.5.8 Conclusion

Cette section a présenté les patrons relationnels MapReduce tels que la sélection, la projection, l'union, l'intersection et la différence. D'autres patrons de conception ont été abordés, comme le groupement, l'agrégation et les jointures. Les problématiques ont été exposées pour illustrer chaque cas et accompagnées des solutions correspondantes, ainsi que le mode d'application de ces solutions.

4.6 Les opérations Trident

Trident est une abstraction de Storm présentant 5 types d'opérations dans le cadre du traitement des flux en temps-réel, partitionné sur un ensemble de nœuds d'un Cluster:

1- Les opérations qui s'appliquent localement à chaque partition sans transfert de données, à travers le Cluster.
2- Les opérations de partitionnement qui découpent les flux en plusieurs partitions réparties sur plusieurs nœuds sans changer le contenu (tout en engageant des transferts de données, à travers le Cluster).
3- Les opérations d'agrégation engageant de par sa nature des transferts de données, à travers le Cluster.
4- Les opérations qui concernent le groupement des flux.
5- Les opérations de fusion et de jointure.

4.6.1 Les opérations locales

Les opérations locales ont lieu dans chaque nœud séparément et n'engagent aucun transfert de données, à travers le Cluster.

4.6.1.1 Les fonctions

Une fonction prend un ensemble de tuples en entrée et en renvoie d'autres ou aucun en sortie. Les données en sortie seront ajoutées aux données en entrée. Dans le cas où il n'y a pas de données en sortie, les données en entrées seront écartées du flux. Sinon, elles seront dupliquées pour chaque donnée en sortie.

Étant donné la fonction suivante s'attaquant à un flux de données en forme de « [a, b, c] » :

```
public class MyFunction extends BaseFunction {
    public void execute(TridentTuple tuple, TridentCollector collector) {
        for (int i = 0; i < tuple.getInteger(0); i++) {
            collector.emit(new Values(i));
        }
    }
}
```

Étant donné également, les données suivantes en provenance du flux :

$[9, 2, 5]$
$[2, 1, 8]$
$[4, 0, 1]$

On exécute alors la fonction « MyFunction » :

```
mystream.each(new Fields("b"), new MyFunction(), new Fields("d")))
```

On arrive par la suite au résultat suivant en sortie :

$[9, 2, 5, 0]$
$[9, 2, 5, 1]$
$[2, 1, 8, 0]$

4.6.1.2 Les filtres

Les filtres prennent les tuples en données d'entrée et décident de les laisser passer en données de sortie ou pas. Étant donnée la fonction « MyFilter » suivante, agissant sur un flux de données en forme de « [a, b, c] » :

```
public class MyFilter extends BaseFilter {
    public boolean isKeep(TridentTuple tuple) {
        return tuple.getInteger(0) == 4 && tuple.getInteger(1) == 7;
    }
}
```

Étant donné également les données suivant en provenance du flux :

[4, 7, 1]
[3, 0, 1]
[4, 1, 9]

On exécute alors la fonction « MyFilter » :

```
mystream.filter(new MyFilter())
```

On arrive par la suite au résultat suivant en sortie :

[4, 7, 1]

4.6.1.3 Les opérations map et flatMap

La fonction « map » standard applique une transformation « 1-1 » des tuples. Par exemple, la fonction « UpperCase » suivante convertit les éléments du flux de données en majuscule :

```
public class UpperCase extends MapFunction {
    @Override
    public Values execute(TridentTuple input) {
        return new Values(input.getString(0).toUpperCase());
    }
}
```

Pour appliquer cette fonction au flux de données, il suffit de lancer la commande :

```
mystream.map(new UpperCase())
```

La fonction « flatMap » est similaire à la fonction « map », seulement elle agit selon une transformation « 1-n » des éléments du flux de données, ainsi qu'un aplatissement des données en sortie dans un nouveau flux.

Étant donné un flux de texte en format phase, on souhaite le convertir en un flux de texte en format mots. Dans ce cas-là, on définit la fonction « FlatMapFunction » suivante :

```
public class Split extends FlatMapFunction {
    @Override
    public Iterable<Values> execute(TridentTuple input) {
        List<Values> valuesList = new ArrayList<>();
        for (String word : input.getString(0).split(" ")) {
            valuesList.add(new Values(word));
        }
        return valuesList;
    }
}
```

On applique alors la fonction « flatMap » au flux de cette manière :

```
mystream.flatMap(new Split())
```

Il est possible également, d'enchaîner différentes opérations de cette manière :

```
mystream.flatMap(new Split()).map(new UpperCase())
```

4.6.1.4 Les opérations min et minBy

Les opérations « min » et « minBy » permettent d'obtenir la valeur minimale dans chacune des partitions du flux Trident. Étant donné un flux de données sous la forme de « [clé, valeur] », réparti sur trois partitions :

$Partition\ 0$:
$[123, 2]$
$[113, 54]$
$[23, 28]$
$[237, 37]$
$[62, 17]$
$[98, 42]$

$Partition\ 1$:
$[64, 18]$
$[72, 54]$
$[2, 28]$
$[742, 71]$
$[62, 12]$
$[19, 174]$

$Partition\ 2$:
$[27, 94]$
$[82, 23]$
$[9, 86]$
$[53, 71]$
$[51, 49]$
$[37, 98]$

Pour obtenir la valeur minimale, il suffit de lancer la commande :

```
mystream.minBy(new Fields("count"))
```

Alors le résultat qui s'affichera est :

$Partition\ 0$:
$[123, 2]$

$Partition\ 1$:
$[62, 12]$

$Partition\ 2$:
$[82, 23]$

4.6.1.5 Les opérations max et maxBy

De la même manière, les fonctions « max » et « maxBy » permettent d'obtenir la valeur maximale dans chacune des valeurs du flux Trident. Dans l'exemple précédent, on lance la commande suivante :

```
mystream.maxBy(new Fields("count"))
```

Ce qui permet d'obtenir les valeurs maximales recherchées :

$Partition\ 0$:
$[113, 54]$

$Partition\ 1$:
$[19, 174]$

$Partition\ 2$:
$[37, 98]$

4.6.1.6 Les opérations de fenêtre

Une fenêtre contient les données d'un flux dans une chronologie et permet d'effectuer diverses opérations sur les données de cette fenêtre. Chaque opération de fenêtre, dite Windowing, génère une valeur de sortie à la fin de la fenêtre. Il existe 3 types de fenêtres :

1- La fenêtre bascule, dite Tumbling Window, permettant de gérer les tuples groupés selon le nombre ou le temps de traitement, sachant que chaque tuple appartient à une fenêtre seulement.
2- La fenêtre glissante, dite Sliding Window, permettant de gérer les tuples groupés pour chaque intervalle glissant, sachant que chaque tuple appartient à une fenêtre seulement.
3- La fenêtre récurrente, dite Common Windowing, fournie par une API configurable, via le paramètre WindowConfig.

La fonction « tumblingWindow » renvoie en sortie un flux de données composé de tuples obtenus d'une agrégation des résultats d'une fenêtre bascule selon la fréquence « windowCount » des tuples.

```
public Stream tumblingWindow(int windowCount,
                             WindowsStoreFactory windowStoreFactory,
                             Fields inputFields,
                             Aggregator aggregator,
                             Fields functionFields);
```

Cette fonction renvoie également en sortie un flux de données composé de tuples obtenus d'une agrégation des résultats d'une fenêtre bascule selon la durée « windowDuration ».

```
public Stream tumblingWindow(BaseWindowedBolt.Duration windowDuration,
                             WindowsStoreFactory windowStoreFactory,
                             Fields inputFields,
                             Aggregator aggregator,
                             Fields functionFields);
```

Par ailleurs, la fonction « slidingWindow » renvoie en sortie un flux de données composé de tuples obtenus d'une agrégation des résultats d'une fenêtre glissante selon la fréquence « windowCount » des tuples.

```
public Stream slidingWindow(int windowCount,
                            int slideCount,
                            WindowsStoreFactory windowStoreFactory,
                            Fields inputFields,
                            Aggregator aggregator,
                            Fields functionFields);
```

Cette fonction renvoie également en sortie un flux de données composé de tuples obtenus d'une agrégation des résultats d'une fenêtre glissante selon la durée « slidingInterval ».

```
public Stream slidingWindow(BaseWindowedBolt.
                            Duration windowDuration,
                            BaseWindowedBolt.Duration slidingInterval,
                            WindowsStoreFactory windowStoreFactory,
                            Fields inputFields,
                            Aggregator aggregator,
                            Fields functionFields);
```

Pour faire appel à l'API des fenêtres récurrentes, c'est bien la fonction « window » :

```
public Stream window(WindowConfig windowConfig,
                     WindowsStoreFactory windowStoreFactory,
                     Fields inputFields,
                     Aggregator aggregator,
                     Fields functionFields) ;
```

4.6.1.7 L'opération partitionAggregate

L'opération « partitionAggregate » lance une fonction sur chaque partition. La seule différence par rapport aux fonctions c'est que les tuples dans les données en sortie remplacent ceux dans les données en entrée.

Étant donné la ligne de commande suivante :

```
mystream.partitionAggregate(new Fields("b"), new Sum(), new Fields("sum"))
```

Étant donné également, un flux de données sous la forme de « ["x", n] » :

$Partition\ 0$:
$["a", 1]$
$["b", 2]$

$Partition\ 1$:
$["a", 3]$
$["c", 8]$

$Partition\ 2$:
$["e", 1]$
$["d", 9]$
$["d", 10]$

En appliquant l'opération « partitionAggregate » sur ces données on obtient en sortie les valeurs suivantes :

$Partition\ 0$:
$[3]$

$Partition\ 1$:
$[11]$

$Partition\ 2$:
$[20]$

Trois interfaces différentes, permettent de définir les agrégateurs :

1- L'interface « CombinerAggregator ».
2- L'interface « ReducerAggregator ».
3- L'interface « Aggregator ».

L'interface « CombinerAggregator » se définit comme suit :

```
public interface CombinerAggregator<T> extends Serializable {
    T init(TridentTuple tuple);
    T combine(T val1, T val2);
    T zero();
}
```

Cette interface envoie un seul tuple en sortie composé d'un champ seulement. Elle lance une fonction « init » sur chaque tulple en entrée, puis combine les valeurs résultats avec la fonction « combine » jusqu'au dernier élément. S'il n'y a pas de tuples dans la partition, cette interface renvoie alors la valeur en sortie de la fonction « zero », comme indiqué dans l'exemple de la classe « Count » suivant :

```
public class Count implements CombinerAggregator<Long> {
    public Long init(TridentTuple tuple) {
        return 1L;
    }

    public Long combine(Long val1, Long val2) {
        return val1 + val2;
    }

    public Long zero() {
        return 0L;
    }
}
```

Les bénéfices de cette interface s'amplifient, en utilisant la fonction « aggregate » à la place de « partitionAggregate ». Dans ce cas, Trident optimisera automatiquement le calcul en faisant une agrégation partielle avant le transfert des tuples à travers le Cluster.

Par ailleurs l'interface « ReducerAggregator » se définit comme suit :

```
public interface ReducerAggregator<T> extends Serializable {
    T init();
    T reduce(T curr, TridentTuple tuple);
}
```

Cette interface produit une valeur initiale avec « init » puis boucle sur cette valeur par chaque tuple en entrée, afin de générer un seul tuple composé d'un seul champ en sortie. Dans notre exemple, la fonction « Count » se définit en tant que « ReducerAggregator » de la façon suivante :

```
public class Count implements ReducerAggregator<Long> {
    public Long init() {
        return 0L;
    }

    public Long reduce(Long curr, TridentTuple tuple) {
        return curr + 1;
    }
}
```

L'interface « ReducerAggregator » peut être utilisée également avec la méthode « persistentAggregate » détaillée dans le paragraphe suivant.

Finalement, l'interface utilisée généralement pour effectuer les agrégations est « Aggregator », définie par :

```
public interface Aggregator<T> extends Operation {
    T init(Object batchId, TridentCollector collector);
    void aggregate(T state, TridentTuple tuple, TridentCollector collector);
    void complete(T state, TridentCollector collector);
}
```

Cette interface est capable de renvoyer en sortie un nombre indéfini de tuples avec un nombre indéfini de champs. Également, les tuples peuvent être renvoyés en sortie à un quelconque moment de l'exécution, qui s'effectue comme ceci :

1- La méthode « init » est appelée avant le traitement. La valeur en sortie de cette fonction est de type Object représentant l'état d'agrégation et sera utilisée en paramètre d'entrée des méthodes « aggregate » et « complete ».
2- La methode « aggregate » est appelée pour chaque tuple en entrée dans la partition. Cette méthode peut éventuellement mettre à jour l'état de l'agrégation et renvoyer des tuples optionnellement.
3- La méthode « complete » est appelée quand tous les tuples dans la partition seront traitée par la fonction « aggregate ».

Dans notre exemple on définit la classe « Count », en tant qu'une interface « Aggregator » comme suit :

```
public class CountAgg extends BaseAggregator<CountState> {
    static class CountState {
        long count = 0;
    }

    public CountState init(Object batchId, TridentCollector collector) {
        return new CountState();
    }

    public void aggregate(CountState state,
                          TridentTuple tuple,
                          TridentCollector collector) {
        state.count+=1;
    }

    public void complete(CountState state,
                         TridentCollector collector) {
        collector.emit(new Values(state.count));
    }
}
```

Dans le cas où on voudrait exécuter plusieurs agrégateurs à la fois, il est possible d'enchaîner les appels aux méthodes correspondant comme ceci :

```
mystream.chainedAgg()
        .partitionAggregate(new Count(), new Fields("count"))
        .partitionAggregate(new Fields("b"), new Sum(), new Fields("sum"))
        .chainEnd() ;
```

Les instructions dans l'exemple précédent permettront de lancer les agrégateurs « Count » et « Sum » dans chaque partition. Les données en sortie sont composées d'un seul tuple sous la forme de « [count, sum] ».

4.6.1.8 L'opération projection

L'opération projection garde seulement les champs spécifiés dans le flux. Étant donné un flux de données sous la forme de « ["a", "b", "c", "d"] », on lance la commande suivante :

```
mystream.project(new Fields("b", "d")) ;
```

Le flux de données en sortie sera alors défini sous la forme de « ["b", "d"] ».

4.6.2 Les opérations de re-partitionnement

Ces opérations lancent une fonction pour changer la manière dont les tuples sont partitionnés à travers les tâches. Par conséquence du re-partitionnement, le nombre de partitions peut changer. Cette manipulation nécessite le transfert des données à travers le réseau. Plusieurs fonctions de re-partitionnement existent :

1- La fonction « shuffle » qui utilise un algorithme RR aléatoire afin de redistribuer les tuples dans toutes les partitions cible.
2- La fonction « broadcast » qui réplique chaque tuple dans toutes les partitions cible.
3- La fonction « partitionBy » qui prend en entrée un ensemble de champs puis effectue un partitionnement sémantique basé sur la variable d'entrée. Les champs sont alors hachés par le nombre des partitions cible. Cette fonction garantie que le même élément en entrée aille toujours à la même partition cible.
4- La fonction « global » assure que tous les tuples soient envoyés à la même partition qui sera utilisée pour tous les Batchs du flux.
5- La fonction « batchGlobal » assure que tous les tubles dans un même Batch soient envoyés à la même partition. Des Batchs différents dans le même flux, utiliseront par conséquent des partitions différentes.
6- La fonction « partition » est une abstraction générique qui implémente la variable « org.apache.storm.grouping.CustomStreamGrouping ».

4.6.3 Les opérations d'agrégation

Trident dispose des deux méthodes « aggregate » et « persistentAggregate » permettant d'effectuer les agrégations dans les flux de données. La méthode « aggregate » est exécutée dans chaque Batch du flux de données séparément, alors que la fonction « persistentAggregate » effectue l'agrégation sur tous les tuples, à travers tous les Batchs du flux de données.

En exécutant la fonction « aggregate » on effectue une agrégation globale sur le flux. Quand on utilise les interfaces « ReducerAggregator » ou « Aggregator », le flux est d'abord re-partitionné dans une seule partition, puis la méthode « aggregate » sera exécutée sur cette partition. En revanche, quand on utilise « CombineAggregator », Trident effectuera des agrégations partielles sur chaque partition, puis une opération de re-partitionnement sur une seule partition, pour finalement finir l'agrégation après le transfert des données, à travers le Cluster. Les interfaces « CombineAggregator » sont beaucoup plus efficaces et doivent être utilisées si possible.

Dans l'exemple suivant, on utilise une agrégation pour obtenir un comptage global du flux :

```
mystream.aggregate(new Count(), new Fields("count")) ;
```

Comme pour « partitionAggregate », les interfaces « Aggregator » avec la méthode « aggregate » peuvent être enchaînées. En revanche, si l'interface « CombinerAggregator » est suivie d'une autre qui n'est pas de type « CombinerAggregator », l'optimisation de l'agrégation partielle par Trident va échouer.

4.6.4 Les opérations correspondant aux flux groupés

L'opération « groupBy » re-partitionne le flux en effectuant une méthode « partitionBy » sur le champ spécifique, puis dans chaque partition, regroupe les tuples ensembles pour ceux parmi eux qui ont les champs groupés à égalité.

Si on lance les méthodes d'agrégation sur un flux groupé, dit GroupedStream, l'agrégation sera exécutée dans chaque groupe, au lieu d'être exécutée sur le Batch entièrement. La méthode « persistentAggregate » peut être également lancée sur un flux groupé. Dans ce cas le résultat est stocké dans une variable « MapState », identifié par les champs groupés ensembles comme clé.

Comme pour les flux réguliers, les méthodes d'agrégation sur un flux groupé peuvent être lancées en chaîne.

4.6.5 Les opérations de fusion et de jointure

La façon la plus simple pour combiner les flux de données est de les fusionner en un seul. Ceci est possible avec la méthode « merge » de Trident. Les noms des champs du nouveau flux seront alors repris des données en sortie du premier flux :

```
topology.merge(stream1, stream2, stream3);
```

L'autre possibilité permettant de combiner les flux de données serait la jointure. Or une jointure standard type SQL nécessite des données finies en entrée, ce qui est impossible avec les données infinies des flux.

Les jointures dans Trident s'appliquent uniquement sur les petit Batchs qui se détachent du traitement principal. Voici un exemple de jointure entre un flux sous la forme de « ["key1", val1, val2] » et un autre sous la forme de « ["key2", val1] » :

```
topology.join(stream1, new Fields("key1"),
              stream2, new Fields("key2"), new Fields("key1", "a", "b", "c"));
```

Cette instruction permet d'effectuer la jointure des 2 flux « stream1 » et « stream2 », en utilisant « key1 » et « key2 » comme champs de jointure respectivement. Ensuite, Trident requiert que tous les champs en sortie du nouveau flux soient nommés. Les tuples résultat de la jointure vont alors contenir :

1- La liste des champs de la jointure. Dans notre cas, « key1 » appartenant à « stream1 » et « key2 » appartenant à « stream2 ».
2- Ensuite, la liste des autres champs des deux flux, non utilisés dans la jointure, dans l'ordre de passation des flux en variable d'entrée. Dans notre cas, « a » et « b » correspondant à « val1 » et « val2 » de « stream1 », puis « c » correspondant à « val1 » de « stream2 ».

A la réalisation de la jointure entre des flux de nature différente, ces derniers seront synchronisés de façon à renvoyer un seul flux en sortie avec les tuples des deux premiers.

4.6.6 Conclusion

Ce paragraphe s'est consacré à quelques primitives fondamentales de Trident. Avec l'API Trident, il est possible de traiter les événements temps-réel en une fois dans la couche dédiée, où :

1- Les tuples sont traités par lots.
2- Chaque lot de tuples reçoit un identifiant unique qui est le numéro de transaction.
3- Si un lot est rejoué, il recevra le même numéro de transaction.
4- Les lots sont traités dans l'ordre.

L'API Trident permet une prise en charge élégante des calculs en temps-réel à travers le traitement des flux, la manipulation des états et les requêtes faible latence, tout en préservant de hautes performances de calcul.

4.7 L'apprentissage automatique et les algorithmes mathématiques

L'apprentissage automatique fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine au sens large d'évoluer grâce à un processus d'apprentissage. Ceci permet d'aborder des problématiques difficiles ou impossibles à traiter par des moyens algorithmiques classiques. Des systèmes complexes peuvent être analysés, y compris pour des données associées à des valeurs symboliques (non pas un nombre mais une probabilité ou un intervalle de définition sur un attribut numérique) ou un ensemble de modalités possibles sur un attribut de valeur (numérique) ou catégoriel. L'analyse peut même concerner des données présentées sous forme de graphes ou d'arbres ou encore de courbes (par exemple courbe d'évolution temporelle d'une mesure). On parle alors de données continues, par opposition aux données discrètes associées à des attributs-valeurs classiques.

4.7.1 Les systèmes d'apprentissage automatique

Le premier stade de l'analyse est celui de la classification, qui vise à étiqueter chaque donnée en l'associant à une classe. Différents systèmes d'apprentissage existent :

1- L'apprentissage supervisé, quand le système apprend à classer selon un modèle de classement prédéterminé.
2- L'apprentissage non-supervisé ou la classification automatique, quand le système ne disposent que d'exemples de données et que l'algorithme doit découvrir par lui-même la structure plus ou moins cachée des données.
3- L'apprentissage semi-supervisé, effectué de manière probabiliste ou pas et visant à faire apparaître la distribution sous-jacente des exemples de données dans leur espace de description.
4- L'apprentissage partiellement supervisé, effectué de manière probabiliste ou pas, quand l'étiquetage des données est partiel.
5- L'apprentissage par renforcement, quand l'algorithme apprend un comportement étant donné une observation.

4.7.2 Les algorithmes d'apprentissage automatique

Les algorithmes utilisés permettent dans une certaine mesure à un système piloté par ordinateurs ou assisté par ordinateur, d'adapter ses analyses et ses comportements en réponse, en se fondant sur l'analyse de données empiriques provenant d'une base de données ou de capteurs. La difficulté réside dans le fait que l'ensemble de tous les comportements possibles compte tenu de toutes les entrées possibles devient rapidement trop complexes à décrire dans les langages de programmation disponibles, de sorte qu'on confie en quelque sorte à des programmes le soin d'apprendre de manière à auto-améliorer le système d'analyse ou de réponse, ce qui est une des formes que peut prendre l'intelligence artificielle.

Ces programmes, selon leur degré de perfectionnement intègrent des capacités en probabilités et statistiques, en traitement des données et éventuellement d'analyse de données issues de capteurs, de reconnaissance (reconnaissance vocale ou reconnaissance de forme), de Data Mining et d'informatique théorique. Parmi les algorithmes utilisés dans ce domaine, on liste :

1- Les machines à vecteur de support.
2- Les réseaux de neurones pour un apprentissage supervisé ou non-supervisé.
3- La méthode des plus proches voisins pour un apprentissage supervisé.
4- Les arbres de décision.
5- Les méthodes statistiques.
6- La régression logistique.
7- L'analyse discriminante linéaire.
8- La logique floue.
9- Les algorithmes génétiques.

Ces méthodes sont souvent combinées pour obtenir diverses variantes d'apprentissage. Le choix de l'algorithme dépend fortement de la tâche à résoudre. Par ailleurs l'apprentissage automatique est utilisé pour un large spectre d'applications :

1- Les moteurs de recherche.
2- L'aide au diagnostic.
3- La bio-informatique.
4- La détection des données aberrantes.
5- La détection des données manquant.
6- La détection de fraudes.
7- L'analyse des marchés financiers.
8- La reconnaissance de la parole.
9- La reconnaissance de l'écriture manuscrite.
10- L'analyse et indexation d'images et de vidéo.
11- La robotique.

4.7.3 Les facteurs de pertinence et d'efficacité

La qualité de l'apprentissage et de l'analyse dépend du besoin en amont et de la compétence de l'opérateur pour préparer l'analyse. Elle dépend aussi de la complexité du modèle et de son adaptation au sujet à traiter. Enfin, la qualité du travail dépendra aussi du mode de mise en évidence visuelle des résultats pour l'utilisateur final (un résultat pertinent pourrait être caché dans un schéma trop complexe ou mal mis en évidence par une représentation graphique inappropriée). Avant cela, la qualité du travail dépendra de facteurs initiaux contraignants liés à la base de données en vigueur :

1- Le nombre d'exemples, sachant que moins il y en a plus l'analyse est difficile mais plus il y en a plus le besoin de mémoire informatique est élevé et plus longue est l'analyse.
2- Le nombre et la qualité des attributs décrivant ces exemples, ce qui signifie que la distance entre deux exemples de données numériques (prix, taille, poids, intensité lumineuse, intensité de bruit) est facile à établir, celle entre deux attributs catégoriels (couleur, beauté, utilité) serait plus délicate.
3- Le pourcentage des données renseignées et manquant.
4- Le bruit, le nombre et la localisation des valeurs douteuses (erreurs) ou naturellement non-conformes au patron de distribution générale des exemples sur leur espace de distribution, impacteront la qualité de l'analyse.

4.7.4 L'apprentissage automatique à l'échelle BigData

L'arrivée de Hadoop a déterminé la constitution d'un écosystème de Frameworks. Rapidement, l'apprentissage automatique a fait son apparition dans cet écosystème, la puissance offerte par Hadoop créant une opportunité pour cela [46] [47].

En effet, les algorithmes d'apprentissage automatique et d'intelligence artificielle en général sont bien souvent très gourmands en ressources de calcul et prennent dès lors beaucoup de temps à s'exécuter. Néanmoins des optimisations sont régulièrement apportées sur ces algorithmes mais elles sacrifient la précision contre le temps [48] [49].

Apache Mahout est une réponse à cette problématique. Il s'agit d'un projet qui n'est pas encore en version finale et consistant en un entrepôt d'algorithmes d'apprentissage automatique, tous conçus pour s'exécuter avec MapReduce. Une vingtaine d'algorithmes sont déjà portés sur le paradigme, ce qui suffit pour la très grande majorité des problèmes.

4.7.4.1 En apprentissage supervisé

Pour ce qui est du contexte de l'apprentissage supervisé, on distingue :

1- La régression logistique visant la prédiction de probabilité d'apparition d'événements, comme par exemple dans le domaine de la fraude bancaire ou encore dans celui de la publicité pour le ciblage de clients à potentiel de fidélité.
2- La classification bayésienne, c'est une application courante sur notre boîte aux lettres électronique nous permettant de filtrer le spam. Les filtres de spam sont en fait des règles de classification qui servent à l'algorithme bayésien contenu dans l'agent anti-spam pour séparer les bons courriels des mauvais.
3- La machine à vecteurs de support, qui est un ensemble de techniques destinées à résoudre des problèmes de discrimination (prédiction d'appartenance à des groupes prédéfinis) et de régression (analyse de la relation d'une variable par rapport à d'autres).
4- Le réseau neuronal qui, à l'inverse des algorithmes de déduction, est un algorithme de type induction. Cela signifie que par le biais d'observations limitées, il essaie de tirer des généralisations plausibles. C'est un système basé sur l'expérience qui se constitue une mémoire lors de sa phase d'apprentissage (qui peut être aussi non-supervisée) que l'on appelle entraînement.
5- Les forêts d'arbres décisionnels, dits Random Forest, qui est une application de graphe en arbres de décision permettant ainsi la modélisation de chaque résultat sur une branche en fonction de choix précédents. On prend ensuite la meilleure décision en fonction des résultats qui suivront. On peut considérer cela comme une forme d'anticipation.
6- Le Boosting qui est une méthode de classification émettant des hypothèses qui sont au départ de confiance faible. Plus une hypothèse est vérifiée, plus son indice de confiance augmente et donc son poids augmente dans la classification.

4.7.4.2 En apprentissage non-supervisé

Pour le contexte d'apprentissage non-supervisé on distingue :

1- KMeans, qui est un algorithme de partitionnement des données en un nombre de groupes K. Dans Mahout on l'utilise typiquement afin de déterminer quelle suggestion peut être faite à un utilisateur en fonction des préférences que d'autres utilisateurs similaires ont eu (utilisateurs du même groupe).
2- Fuzzy KMeans, s'agissant d'une variante du précédent algorithme proposant qu'un objet ne soit associé qu'à un seul groupe.
3- L'algorithme EM, utilisant des probabilités pour décrire qu'un objet appartient à un groupe. Le centre du groupe est ensuite recalculé par rapport à la moyenne des probabilités de chaque objet du groupe.
4- Le regroupement hiérarchique qui a deux sous-algorithmes dérivés. Le premier, Bottom Up a pour fonction d'agglomérer des groupes similaires donc en réduire le nombre et d'en proposer un ordre hiérarchique. Le deuxième, Top Down fait le raisonnement inverse en divisant le premier groupe récursivement en sous-ensembles.

4.7.5 Conclusion

L'apprentissage automatique peut s'appliquer à des volumes élevés de données afin d'obtenir une compréhension plus fine des processus en œuvre, par conséquent, améliorer la prise de décision dans la fabrication, la vente au détail, la santé et les sciences de la vie, le secteur du tourisme et de l'hospitalité, les services financiers et l'énergie, les matières premières et les services publics. Les systèmes d'apprentissage automatique, peuvent faire des prédictions de résultat précis basé sur des données issues de formation ou d'expériences passées. En rassemblant des informations pertinentes pour une prise de décision plus précise, les systèmes d'apprentissage machine peuvent aider les fabricants à améliorer leurs opérations et de leur compétitivité.

Bien que les techniques d'apprentissage automatique aient largement augmenté leur niveau d'applicabilité et la pertinence des scénarios dans le monde réel, quelques défis majeurs perdurent lorsqu'il s'agit de leur mise en œuvre. Certains d'entre eux sont:

1- Le manque d'expertise dans l'application des techniques d'apprentissage machine aux problèmes des entreprises.
2- Le manque de connaissance pour appliquer le processus d'apprentissage machine à des opérations quotidiennes.
3- La disponibilité des bonnes données provenant de diverses opérations et processus.
4- Le manque de compétence technologique dans l'utilisation de BigData pour les algorithmes d'apprentissage automatique.

4.8 Conclusion du chapitre

Les modèles de données fournissent un aspect visuel permettant la gestion des ressources de données et la création des architectures fondamentales, de façon à mieux optimiser l'usage des données des applications tout en réduisant les coûts de traitement. Toutes les techniques de modélisation disposent de points forts et moins forts dans leur façon d'aborder l'utilisateur. La majorité des techniques disponibles est adaptée à un fonctionnement développeur, alors que très peu sont dédiées aux utilisateurs ayant des connaissances techniques limitées. De plus, ces techniques produisent des modèles de données complexes en tentant de couvrir tous les aspects, au détriment de la simplicité d'usage de la solution et la lisibilité des données. Le modèle le plus adapté est le résultat d'une combinaison entre l'efficacité d'un point de vue technique et la simplicité et la convivialité de l'usage du modèle.

Partie 2. La modélisation intégratrice du traitement BigData

Chapitre 5. Le pré-traitement par étude de cas

5.1 Introduction au chapitre

Dans le chapitre précédent on a expliqué plusieurs patrons et algorithmes MapReduce dans le but de fournir une vue systématique des différentes techniques disponibles. On a illustré la théorie par des cas d'emploi, notamment pour les patrons basiques, non-basiques et relationnels, ainsi que par l'API Trident et l'apprentissage automatique. Pour tous ces exemples, on a fourni les descriptions et les codes sources du modèle standard de Hadoop MapReduce.

Ce chapitre expose un modèle dynamique pouvant être appliqué dans le traitement des données volumineuses. Les fondations de cette étude seront décrites en détail, en commençant par l'acquisition et le stockage des données, l'identification des structures et des types de données et l'établissement des liaisons entre les entités. Ensuite les procédures d'utilisation des systèmes experts seront définies afin d'améliorer les performances de traitement des données. Pour finir, ces concepts seront appliqués dans le domaine du traitement des données en entreprise, ainsi que dans le domaine des réseaux sociaux.

5.1.1 Les idées de départ

Le défi du traitement BigData se résume à l'interrogation rapide des données dans la base non-relationnelle. Pour ce faire, on va définir un nouveau modèle basé sur les aspects suivants :

1- L'acquisition et le stockage des données.
2- L'établissement des liaisons entre les entités.
3- L'identification des structures et des types de données.

Ensuite, on va démontrer l'amélioration des performances de traitement à l'usage des systèmes experts, puis exposer des cas d'emploi très classiques :

1- La gestion des profils des revendeurs.
2- L'optimisation du trafic routier.
3- La prédiction des taux de désabonnements.

5.1.2 Le format JSON

Les modèles de données orientés document structurent les données dans des formats standards, tels que XML, YAML, JSON et BSON, ainsi que des formats binaires, tels que PDF ou les formats Microsoft Office (ancien format).

Certains formats de fichiers comme XML permettent de définir un schéma de différentes façons, alors que d'autres comme JSON ne disposent pas de schéma accompagnant les données. En revanche, leur schéma est défini implicitement dans le contenu du fichier. Il suffit donc de déduire le schéma de données à partir d'un échantillon de fichier afin de fournir le modèle d'accès à tout autre fichier de même type et conforme au schéma déduit.

JSON est la technique la plus utilisée pour échanger les données sur Internet. En se basant sur le contenu du fichier, on pourrait déduire un schéma composé éventuellement de trois groupes :

1- Les valeurs, qui sont le plus bas niveau de données. Elles peuvent être de type chaîne de caractères, nombres entier ou décimal, binaire, date et même de type objet et tableau.
2- Les objets, contenant un ensemble de couples clé-valeur.
3- Les tableaux, qui sont des listes d'objets.

Les fichiers JSON sont compatibles avec tous les langages de programmation et très connus des développeurs. Leur format compatible avec Javascript fait en sorte qu'ils soient aussi fortement utilisés dans les échanges de données sur Internet et présents à la fois sur les ordinateurs et les terminaux mobiles. Un exemple de fichier JSON correspondant à des données météo est illustré dans la Figure 46.

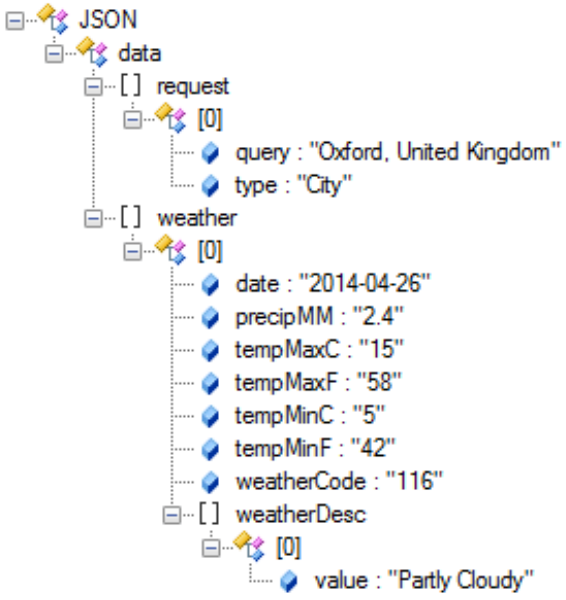

Figure 46 : Schéma implicite des données météo de type JSON

5.1.3 Le schéma de données implicite

La manipulation de données JSON même complexes et y compris en quantités importantes est facilement supportée par les outils logiciels à notre disposition. L'extraction de données de type JSON à partir des API disponibles sur Internet peut exposer une complexité arbitraire de ces données. L'option « Path Constructor » permet d'extraire et de structurer les données depuis chaque objet dans des tableaux. Chaque instance est reconstruite sous forme d'un enregistrement dans la table de données correspondante au sein du Dataset.

Dans certains formats on utilise des structures imbriquées pour simplifier le groupement des données. Il est conseillé d'aplatir ces structures pour qu'elles soient disponibles sous forme de colonnes dans les tables de données. L'option « Subpath Constructor » peut être utilisée dans le Dataset pour intégrer ces structures imbriquées, au moment de l'aplatissement de l'objet racine ou les données sélectionnées, via l'option « Path Constructor ». La puissance des structures imbriquées apparaît au moment de l'usage de la table de données parente dans un contexte récursif. A ce moment précis, toute référence de données dans la structure imbriquée est synchronisée avec l'enregistrement de données en cours de traitement dans la boucle. Durant la création des Datasets à partir du contenu des fichiers JSON, il est capital d'établir les liaisons de données correctement. Les clés primaires doivent être également créées en utilisant les identifiants de chaque nœud, son niveau dans la hiérarchie, sa structure et son comportement [50]. Dans notre exemple de données météo, le Dataset correspondant est décrit dans la Figure 47.

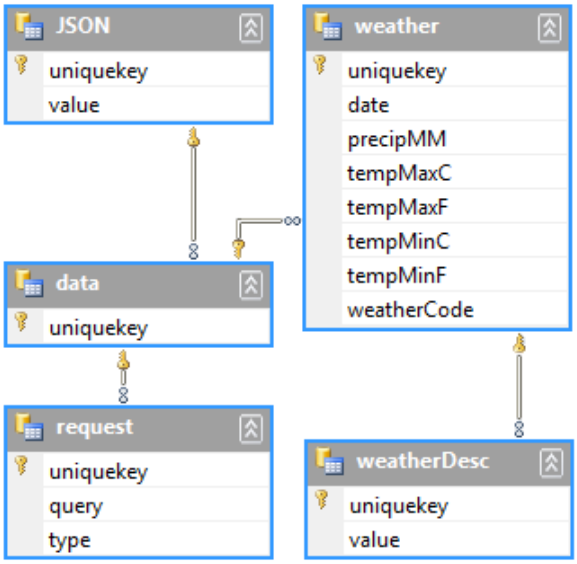

Figure 47 : Dataset correspondant aux données météo de type JSON

5.1.4 Le concept de pré-traitement

Dans la plupart des cas le fichier initial est utilisé seulement comme un échantillon de données permettant de déduire le schéma implicitement. Il est possible également de charger un ou plusieurs autres fichiers sélectionnés au hasard lors de l'exécution pour une validation préalable du schéma de données [51] [52]. Ainsi, un contrôle qualité sera fait et permettra d'identifier les branches introuvables. A défaut, si lors de l'exécution un fichier de structure différente même légèrement est rencontré, la lecture des données va échouer.

Après validation du schéma de données, le processus de lecture sera prêt à l'exécution afin d'extraire les données à partir d'une API de type JSON et de les stocker dans la base. Ces données seront utilisées par la suite dans le moteur d'inférence recherché. Dans les paragraphes qui suivent, on va expliquer comment créer un modèle de prédicats utilisé dans un pré-traitement des données BigData, avec des résultats performants selon des études similaires [53] [54].

Le concept de pré-traitement est illustré dans la Figure 48.

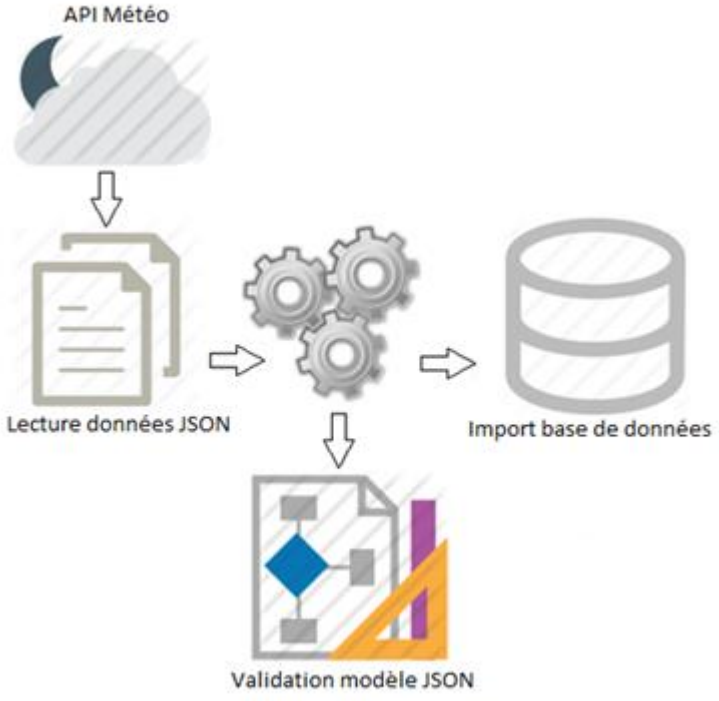

Figure 48 : Pré-traitement des données météo de type JSON

5.2 Les systèmes experts

D'après la définition de Feigenbaum en 1982, un système expert est un logiciel intelligent qui utilise des connaissances et des inférences logiques pour résoudre des problèmes qui sont suffisamment difficiles pour nécessiter une expertise humaine important pour trouver une solution. En d'autres termes, un système expert est un logiciel qui sait donner des recommandations pour un périmètre d'application bien défini, au même niveau d'un expert humain de ce périmètre. Ces recommandations se traduisent par un mode de raisonnement et s'expriment par des règles telles que : « SI telle condition est vérifiée ALORS effectuer telle action » (modus ponens). A partir des bases de connaissance contenant soit l'expertise de spécialistes, soit les connaissances implicites que l'homme utilise en permanence (telles que la compréhension du langage ou la reconnaissance de formes), de nouvelles applications cherchent aujourd'hui à simuler le raisonnement et les comportements humains.

5.2.1 Les notions de base des systèmes experts

Les SE ont commencé à intéresser les industriels il y a environ 20 ans avec l'arrivée des premières réalisations opérationnelles. Les intérêts de chacun sont différents et correspondent à des besoins précis d'une entreprise ou d'un service. En effet chaque réalisation d'un SE s'intègre dans un environnement particulier de développement et doit être adaptée au cadre d'utilisation. En règle générale, un SE est développé dans le but de :

1- Sauvegarder une expertise accumulée bien souvent dans une entreprise et avant le départ de l'expert disposant du savoir-faire requis.
2- La diffuser dans le temps dans le but d'apporter une amélioration ou une évolution accompagnant le changement des processus de l'entreprise.
3- La diffuser dans l'espace de façon à ce que plusieurs contributeurs non-spécialistes puissent y accéder sans contraintes techniques particuliers.
4- Formaliser une connaissance de conception dans le cas des guides de dépannages étape-par-étape des appareils électriques et électroniques.

5.2.1.1 L'architecture des systèmes experts

Un SE est composé de deux modules : la base de connaissance et le moteur d'inférence :

1- La base de connaissance contient les faits de savoir-faire et les règles de l'expert. Elle intègre deux types de faits : les faits permanents du domaine et les faits déduits par le moteur d'inférence.
2- Le moteur d'inférence est un programme chargé d'exploiter la BC pour mener un raisonnement sur le problème posé en fonction du contenu de la base de faits. Pour cela, il contient un algorithme qui examine les conditions de règles et vérifie si elles sont vraies ou fausses. Une règle dont la prémisse (ou partie condition) est vraie est considérée comme applicable. Une prémisse peut contenir une ou plusieurs conditions. Chaque condition correspond à un fait : elle est vraie si le fait est présent dans la base, fausse si le fait contraire est présent et inconnue si le fait est absent. Chaque règle possède également une conclusion. Ce peut être une action à effectuer ou un fait à déduire et à rajouter dans la base de faits.

5.2.1.2 Le fonctionnement des systèmes experts

Le moteur d'inférences se charge de détecter les règles applicables, de choisir parmi elles celle qu'il convient d'appliquer et finalement de l'exécuter. La réalisation du SE et son utilisation posent donc tous les problèmes classiques rencontrés en informatique :

1- La connaissance et l'analyse du domaine, des besoins et de l'environnement d'utilisation requièrent un soin particulier. Il est très important que les limites d'action du SE soient bien fixées.
2- Les éléments de travail devront être judicieusement choisis car le bon déroulement d'un projet SE repose sur une collaboration importante entre les différents acteurs : experts, informaticiens et responsables.
3- Définir la responsabilité relative à l'expertise une fois informatisée. En effet, dans le cas d'un SE de diagnostic de panne comme dans le cas d'un système de contrôle / commande où un arrêt peut coûter cher, qui doit être responsable d'une erreur dans le programme, l'informaticien qui a conçu le SE, l'expert qui a fourni la connaissance ou le cogniticien qui l'a analysée ?

5.2.1.3 L'apprentissage des systèmes experts

L'efficacité et la convivialité des SE dépend fortement des modules, fonctions, interfaces, logiciels, qui gravitent autour du cœur du SE (BC + moteur d'inférence). L'apprentissage dans le domaine des SE est une amélioration automatique des Bases de Connaissances (BC) afin d'améliorer les résultats d'expertise fournis. Ceci signifie deux choses :

1- Modifier les règles de la BC, à partir d'un jugement porté par l'expert sur les résultats.
2- Elaborer de nouvelles règles par généralisation inductive à partir de cas individuels.

Par ailleurs, un SE qui n'est pas doté d'un module d'apprentissage aura naturellement le même comportement dans les mêmes conditions. En d'autres termes, il reproduira les mêmes erreurs si on lui fournit les mêmes entrées.

Finalement, les principales difficultés que l'on rencontre au cours du développement d'un SE se trouvent au niveau de l'expression de la connaissance. Les problèmes interviennent dès l'analyse du discours recueilli chez l'expert. L'acquisition des connaissances de l'expert est toujours une tâche longue et fastidieuse. Elle met en collaboration deux personnes : l'expert et le cogniticien, sur lesquelles repose la construction du SE. Son succès dépend fortement des rapports que ces deux intervenants auront entretenu ensemble.

Alternativement, la question de la cohérence de la base de règles est NP-complète, en d'autres termes nous ne disposons pas d'algorithme économique pour contrôler cette cohérence et donc le déterminisme du système. Ceci n'est pas rédhibitoire pour un petit nombre de règles, mais devient éventuellement problématique sur des cas complexes

5.2.1.4 La déduction naturelle

La déduction naturelle est simplement la pratique du raisonnement en pas élémentaires évidents. Les règles de déduction naturelle constituent les raisonnements élémentaires autorisés. Chaque règle est constituée de formules dites hypothèses et d'une conclusion.

Deux familles de règles existent en général :

1- Les règles d'introduction permettant de générer un symbole de la logique afin de bâtir progressivement la formule à prouver.
2- Les règles d'élimination servent à simplifier un raisonnement ou à obtenir une nouvelle formule grâce aux raisonnements déjà réalisés et ainsi faire disparaître un connecteur logique.

Modus Ponens et Modus Tollens sont deux types d'inférences pouvant être tirées à partir d'une proposition hypothétique « SI a, ALORS b ». Modus Ponens ou détachement se réfère aux inférences de la forme « SI a, ALORS b » et poser ensuite l'antécédent « ET a » pour en déduire le conséquent « DONC b ». Modus Tollens ou contraposition se réfère aux inférences de la forme « SI a, ALORS b » et poser ensuite la négation du conséquent « OR NON(b) » pour en déduire la négation de l'antécédent « DONC NON(a) ». La contraposition est une règle dérivée du détachement. Ce qui signifie que la proposition contraposée de la proposition « SI a, ALORS b » serait « SI NON(a), ALORS NON(b) »

5.2.1.5 Le raisonnement par étude de cas

Le raisonnement par étude de cas dit Case-Based Reasoning, est un paradigme de résolution de problèmes qui cherche à résoudre un problème cible en s'appuyant sur une base de cas passés résolus. Un cas est constitué d'un problème source et de sa solution source associée. Le CBR peut être modélisé par un cycle constitué de quatre étapes : élaboration, recherche, réutilisation, révision et apprentissage. Ces étapes gravitent autour d'une base de connaissances du domaine d'application. Chacune des étapes du cycle mobilise ces connaissances pour supporter la recherche de la solution du problème cible :

1- La première étape du CBR est de construire la spécification du problème à résoudre. À partir d'une requête initiale soumise au système, des informations et connaissances sont inférées afin de mieux orienter la recherche pour un objectif et une tâche précis.
2- Le problème cible élaboré est ensuite utilisé dans l'étape de recherche pour retrouver un cas similaire dans la BC. Déterminer la similarité entre deux cas implique l'utilisation de mesures et de connaissances de similarités fortement liées au domaine d'application.
3- Durant l'étape de réutilisation, la solution du cas source est adaptée pour obtenir une solution au problème cible. L'adaptation est un des processus les plus délicats du CBR.
4- Si la recherche est faite en anticipant correctement sur la réutilisation, cette dernière en sera d'autant plus facilitée et l'on sera sûr d'avoir un cas adaptable. Les connaissances de similarité et d'adaptation apparaissent alors comme étant duales voire même confondues. C'est lors de la phase de révision que la solution proposée peut être corrigée, acceptée ou refusée par l'utilisateur. Cette étape permet d'évaluer l'adaptation. Elle permet également de préparer l'apprentissage puisqu'elle fait émerger de nouvelles connaissances. L'étape de révision permet également d'évaluer l'utilité du cas nouvellement résolu et d'élaborer une stratégie de rétention ou d'oubli des cas selon leur contribution à la compétence du système.

5.2.2 La connexion SGBD et SE

Il est intéressant de connecter les SGBD et les SE ensemble. Les SGBD fournissent des facilités bas-niveau, en revanche, ils assurent une assistance minime en termes d'interface utilisateur et d'extraction de données. Les SGBD ne permettent pas de faire des raisonnements logiques à partir des données stockées, ce qui empêche de mettre en avant la valeur intrinsèque des données. Par conséquent, le SGBD couplé à un traitement par inférence est plus performant et plus efficace.

Les SGBD et les SE peuvent fonctionner comme des collaborateurs indépendants. Le SGBD peut être étendu, en y ajoutant des règles et des capacités de traitement. Pour ce faire, il est important d'identifier des prédicats dans la base de données, correspondant aux liaisons dans le SGBD. Lors du traitement d'une requête, les prédicats sont convertis en requêtes du SGBD. Au moment de l'envoi de la requête, une relation résultat est utilisée pour compléter le traitement.

Étant donné un objectif « O1 » et un sous-objectif de la fonction « f(a, b) » de la base de données, les valeurs « a » et « b » renvoyées par le SGBD sont liées aux instances de « a » et de « b » dans « O1 ». L'action permettant de dériver l'objectif « O2 » de l'objectif « O1 » par substitution et suppression du sous-objectif résolu correspond à l'étape d'inférence dite Modus Ponens. Pour chaque tuple renvoyé par la base de données, il existe une étape d'inférence associée, qui doit être maintenu de façon cohérente avec la stratégie de recherche en profondeur. La première étape d'inférence est effectuée avec le premier tuple renvoyé. Les autres tuples sont utilisés à chaque fois l'objectif « O1 » est sollicité, en tant que résultat de la phase du retour sur trace, dit Backtracking. Afin d'obtenir de meilleures performances, il est important d'éviter d'accéder à la base de données à chaque fois qu'un prédicat est traité, en tant que sous-objectif. La meilleure approche décalerait l'évaluation de ces prédicats, en attendant leur conversion en requêtes du SGBD. Par conséquent, la communication entre le SGBD et le SE sera réduite.

Globalement, on constate qu'il est plus intéressant en termes de performance, d'introduire un pré-traitement des données via le moteur d'inférence et avant envoi au serveur de calcul Hadoop. Dans ce type d'implémentation, la couche logicielle simule le moteur d'inférence et chacune de ses étapes. Les prédicats dans la base de données seront alors passés jusqu'à ce que tous les non-prédicats soient éliminés. Ce type de moteur d'inférence méta-niveau [55] mène ses interactions à travers la base de données, en utilisant les capacités en entrée / sortie fournies. Le résultat final est reconstruit à partir des valeurs renvoyées par le SGBD. Ce fonctionnement est illustré dans la Figure 49.

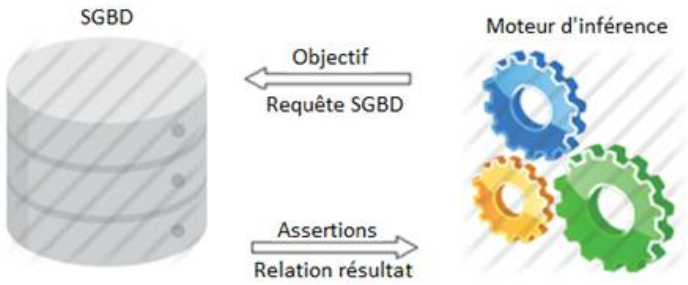

Figure 49 : Connexion SGBD et SE

Cette connexion peut être implémentée alternativement en faisant une modification directe du moteur d'inférence afin de pouvoir accomplir les tâches correspondant au méta-niveau. Cette méthode est moins portable mais elle est plus efficace qu'un méta-niveau logiciel. Cependant, dans le cas de la création d'un nouveau système offrant des fonctionnalités étendues, cela reste le meilleur scénario à appliquer.

5.2.3 Les règles de traitement

En utilisant une syntaxe de règles on peut implémenter un ensemble de capacités SGBD supplémentaires. Ces capacités peuvent être des vues, des triggers, des contraintes d'intégrité et des services de protection de la base de données. Ces règles peuvent être utilisées séparément (implémentées par des mécanismes indépendants) ou combinées ensembles dans un Framework seul et unique embarqué dans l'outil de traitement des requêtes du SGBD.

5.2.3.1 Les vues de données

Elles sont créées par une règle permettant aux opérations utilisateur sur une liaison de vue de devenir une des opérations actuelles sur les liaisons associées, stockées dans la base de données.

5.2.3.2 Les indicateurs de performance de la base de données

Ils subissent une surveillance permanente par les déclencheurs, dits Triggers. Si ces indicateurs s'affichent positivement, les actions correspondant, permettant de modifier la structure de la base de données ou l'envoi de courts messages, seront exécutées.

5.2.3.3 Les services de protection de la base de données

Ils limitent les accès individuels à des endroits spécifiques de la base de données.

5.2.3.4 Les contraintes d'intégrité

Elles protègent l'état de la base de données pendant les mises-à-jour. Si une ou plusieurs contraintes ne sont pas respectées, la mise-à-jour de la base de données sera alors rejetée.

5.2.4 Le moteur d'inférence

En plus des capacités classiques d'un SGBD, il est possible également de mettre en place un système de règles personnalisé dans une base de données secondaire de faits et de règles. Cette approche consiste à créer un ensemble de tables logiques de base de données qui, combinées ensembles pendant le traitement, contrôleront le résultat de calcul en se basant sur des modèles spécifiques à définir. Ces règles peuvent physiquement exister dans une base de données ou configurées dans un fichier script externe. Il existe quatre groupes principaux à gérer [56].

5.2.4.1 Les profils de mesures

Les profils de mesures dits Profile Measures sont utilisés pour relever un ensemble de d'indicateurs KPI différents, à identifier dans les données principales. Pour chaque profil, il doit être défini un type de données et des valeurs maximales / minimales dans le cas de données numériques.

Le Tableau 3 est une illustration de quelques modèles de prédicats définis dans la base de données. Chaque profil numérique se situe entre une valeur maximale et une valeur minimale.

Identifiant	Libellé	Type	Maximum	Minimum
1	Profil 1	Texte	AAA	
2	Profil 2	Numérique	0	100
3	Profil 3	Numérique	-20	10
4	Profil 4	Booléen	Faux	Vrai
...	...	…	...	...

Tableau 3 : Définition des profils dans le pré-traitement par étude de cas

5.2.4.2 Le filtre par modèle

Le filtre par modèle dit Pattern Matching permet de retrouver le modèle de prédicat le plus adapté aux données. Cette identification se base sur un ou plusieurs profils de mesures spécifiques disponibles dans les données principales. Si plusieurs modèles de prédicat sont sélectionnées, il sera retenu le modèle le plus avancé comportant des indicateurs KPI détaillés. Pour chaque modèle, il doit être défini une règle sous la forme d'une chaîne d'une ou plusieurs opérations logiques et composées des identifiants des profils.

Le Tableau 4 est une illustration de quelques modèles de prédicats définis dans la base de données. Chaque règle est composée d'une équation logique permettant de valider une certaine combinaison de profils par leur identifiants.

Identifiant	Libellé	Règle
1	Modèle 1	#9 == false && #14 && #49 ...
2	Modèle 2	#7 > 0 && #8 > 0 && #31
3	Modèle 3	(#14 || #19) && (#49 || #2...
4	Modèle 4	#14 && #49 && #19 == ...
...	...	...

Tableau 4 : Définition des modèles de prédicats dans le pré-traitement par étude de cas

5.2.4.3 Les règles de cas ou de conditions

Les règles de cas ou de conditions dites Condition Rules sont utilisées pour contrôler les données et s'assurer qu'elles correspondent à un ou plusieurs critères du modèle de prédicat. Chacune de ces conditions est définie sous forme d'une règle qui calcule le résultat basé sur les valeurs de profils correspondant dans les données d'entrée. Il y a deux types de conditions, les primaires et les secondaires, sachant que les conditions primaires définissent une sorte d'agrégation des valeurs secondaires.

Le Tableau 5 est une illustration de quelques règles de cas ou conditions permettant d'évaluer le niveau d'application du modèle choisi en évaluant la valeur des profils par leurs identifiants. La particularité de cette étape se situe au niveau de la colonne Primaire. Cette colonne spécifie s'il s'agit d'une condition primaire, permettant de calculer un sous-total de la valeur cherchée ou pas. Si c'est le cas, alors les identifiants de l'équation dans la colonne Règle seront les identifiants des autres conditions qui servent à calculer le sous-total et non pas les identifiants des profils, permettant d'évaluer une condition. La logique de cette démarche est bien entendue gérée par la couche applicative.

Identifiant	Libellé	Règle	Primaire
1	Condition 1	#39	Vrai
6	Condition 2	#16 < 1 ? 0 : (#16 < 6 ? …	Faux
9	Condition 3	#17 < 1 ? 0 : (#17 < 11 …	Faux
11	Condition 4	#18	Faux
13	Condition 5	#14 + #15 + #16 + #17 …	Vrai
14	Condition 6	#20 == Vrai ? 0.5 : 0	Faux
...	...	...	...

Tableau 5 : Règles de cas ou de conditions dans le pré-traitement par étude de cas

5.2.4.4 Les relations résultats

Les relations résultats dites Result Relations existent sous forme de règles utilisées pour appliquer une pondération aux résultats précédents. Ces règles prennent le résultat de calcul des conditions comme données d'entrée et fournissent le résultat final après pondération. Comme pour les règles de conditions, les relations résultats utilisent aussi les agrégations et les combinaisons logiques.

Le Tableau 6 est une illustration de quelques règles de pondération permettant d'appliquer un niveau de pondération sur les résultats des calculs précédents. Chaque règle est évaluée en fonction des identifiants des conditions primaires dans la phase précédentes. De la même manière que les règles de cas, la colonne Primaire permet de calculer un total basé sur un ensemble de règles de pondération par leur identifiants. L'ensemble des totaux est regroupé à la fin pour indiquer le niveau de réalisation du modèle de prédicat sélectionné au départ.

Identifiant	Libellé	Règle	Primaire
1	Total 1	#47	Vrai
3	Sous-total 1	(#48 * #25) / 100	Faux
4	Total 2	#49	Vrai
8	Sous-total 2	(#55 * #8)	Faux
11	Sous-total 4	(#56 * #13)	Faux
16	Total 3	#50	Vrai
19	Sous-total 6	(#57 * #31)	Faux
...	...	...	...

Tableau 6 : Relations résultats dans le pré-traitement par étude de cas

5.2.5 Le pré-traitement par étude de cas

Les calculs nécessitant un temps de traitement considérable en finance, BI ou tout autre domaine d'activité peuvent être traités désormais sur les serveurs du Cloud. Des plate-formes intuitives sont disponibles publiquement afin de pouvoir lancer des traitements des données coûteux en termes de temps de calcul sur des réseaux de serveurs comportant un nombre important de processeurs. Seules les grandes entreprises et les universités avaient accès jusqu'à maintenant à des serveurs de traitement de hautes performances. Ce fait menait à des inconvénients compétitifs importants pour les petites et moyennes entreprises. En utilisant les solutions actuelles des fournisseurs, les serveurs de traitement de hautes performances sont accessibles à tous. Les utilisateurs étant facturés pour ce qu'ils consomment en temps de traitement, les réseaux de serveurs évolutifs permettent de réduire les coûts en dehors des heures de travail.

Afin de baisser les coûts de traitement davantage, il est important d'envoyer sur les serveurs de calcul seulement les données indispensables pour obtenir le résultat final. Les moteurs d'inférence et les modèles de prédicats permettent de faire cette étape de préparation. Le pré-traitement, via un raisonnement par étude de cas, analyse les données en amont et fournit une solution basée sur une sélection des opérateurs à utiliser après comparaison avec la bibliothèque de cas.

Cette technique permet d'automatiser le travail manuel fait par un développeur dans le cas classique. Le traitement CBR simule le comportement humain dans la résolution des problèmes informatiques, en essayant de retrouver des cas analogues dans sa base de connaissances et en appliquant une adaptation au cas en cours. Cette technologie remonte à 1977 mais elle reste très peu connue par rapport à d'autres dans le même domaine comme le Data Mining.

Étant donné « E » l'ensemble des données, qu'elles proviennent de la source de données ou qu'elles soient des résultats de calcul, puis « O », l'ensemble des opérateurs disponibles et enfin « Os $\subset$ O », l'ensemble trié (afin d'obtenir de meilleurs résultats) des opérateurs sélectionnés. On définit alors :

$o \quad Os$
$d \quad E$
$r \quad E$
$r = o(d)$

On définit également :

$r' \quad E$
$r' = o(r)$

Le CBR lance le flux et propose à l'utilisateur la ou les solutions qui correspondent au changement de structure. Le CBR alerte alors l'utilisateur afin d'intervenir et de faire un nouveau Mapping des données. Ainsi il permet à l'utilisateur d'enrichir la base de cas en y rajoutant sa solution. Dans le chapitre qui suit on va illustrer ces propos par des cas d'emploi différents dans le cadre de l'expérimentation réalisée.

5.2.6 L'apprentissage automatique

L'apprentissage automatique est la dernière étape du processus relative à l'enrichissement de la base de cas, en y ajoutant l'adaptation du cas appliqué ou la résolution survenue après intervention d'un expert, faute de ne pas trouver un cas similaire dans la base. Il est en revanche non nécessaire de rajouter à la base de cas, une solution générée d'un cas précédemment connu et sans adaptation. Ce processus est illustré dans la Figure 50.

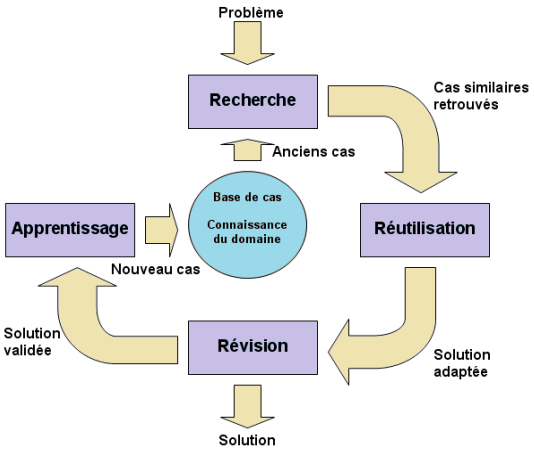

Figure 50 : Processus de résolution du CBR

5.2.7 Conclusion

Plusieurs outils et algorithmes sophistiqués ont été développés pour une meilleure gestion du traitement des données et de nombreuses recherches de bonne qualité ont été accomplies dans ce domaine. Ces travaux se focalisent toujours sur l'amélioration d'un traitement des données en une fois. Le présent chapitre envisage un pré-traitement à base d'une modélisation intégratrice, qui s'attaque aux données en amont, avant le traitement définitif sur le Cloud, en utilisant un ensemble d'opérateurs et de techniques de modélisation, ainsi qu'un pré-traitement via un raisonnement par étude de cas.

Le modèle proposé dans les paragraphes précédents permet de faire une extension au SGBD standard reposant sur un ensemble de règles prédéfinies. Les principaux groupes affichés précédemment permettent d'améliorer les fonctions de notation de profil [57], en utilisant l'inférence pendant un traitement des données en local. Dans le paragraphe suivant, on va expliquer comment adopter ce modèle dans une échelle de données macro, en faisant un traitement des fichiers de données sur les nœuds et en utilisant les règles et prédicats du moteur d'inférence. Cette étude illustrée dans la Figure 51, montre une amélioration significative en termes de performances de calcul.

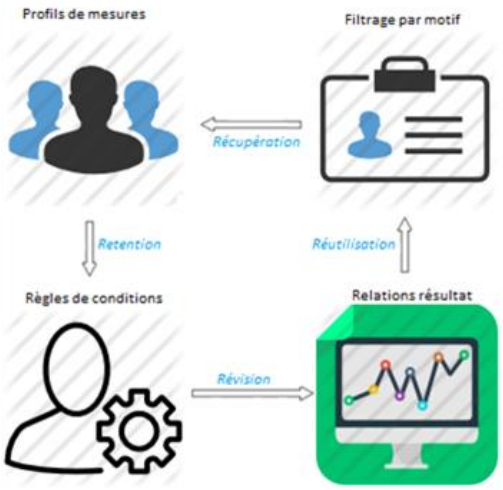

Figure 51 : Exemple d'implémentation d'un moteur d'inférence pour pré-traitement

5.3 La surveillance des réseaux sociaux

Les réseaux sociaux de nos jours font l'objet d'une croissance significative au niveau mondial. Ils ont changé le mode de fonctionnement de l'économie et de la société. Pour les entreprises et les marques, le canal des réseaux sociaux prend de plus en plus, une place importante dans sa façon d'interagir avec leurs objectifs. La croissance énorme de l'utilisation des réseaux sociaux, a développé ses propres dynamiques. Une brève publication peut se transformer en un incident critique pour les organisations, les entreprises et les marques.

De ce fait des outils puissants en termes de surveillance et utilisant un modèle optimisé de traitement des données, sont absolument nécessaires. Dans ce chapitre, on applique la modélisation intégratrice à trois étapes : acquisition, modélisation et traitement des données. On utilise des opérateurs de modélisation implémentant le raisonnement par étude de cas. Cette boîte à outils permet d'obtenir des résultats intéressants en termes de traitement des données et d'informatique écologique, dit Green Computing.

5.3.1 La nature et les avantages des réseaux sociaux

Les réseaux sociaux sont considérés comme un outil de partage de données entre utilisateurs intéressés. Ils ont fait évoluer le flux classique des données, auparavant en mode poussé, dit Push, vers un mode plus interactif dans les deux sens, dit Push-Pull. De nos jours, la communication d'une entreprise est considérée comme une parmi de nombreuses sources d'information en termes de comportement international, opportunités de travail, produits et services.

Depuis quelques années, les consommateurs et tout autre type d'utilisateurs pouvaient consulter le site Internet de la marque pour avoir des informations concernant les produits et les services. De nos jours, la source d'information principale est devenue le canal des réseaux sociaux, permettant davantage d'interactivité et de dialogue. La Figure 52 illustre la différence entre le flux d'information sociale d'hier et celui d'aujourd'hui.

Figure 52 : Flux d'information sociale entre hier et aujourd'hui

Les réseaux sociaux offrent de nombreuses possibilités pour une marque [58], dont voici quelques avantages :

1- Transporter l'image de marque et celle de l'entreprise à toute une autre dimension.
2- Encourager les consommateurs pour recommander directement ou indirectement les produits et service de la marque sur leurs réseaux sociaux.
3- Établir une relation à long-terme avec les consommateurs.
4- Améliorer la qualité de service via la remontée directe des consommateurs, dite Feedback.
5- Améliorer les produits à travers la connaissance pertinente du marché.

5.3.1.1 Les catégories des réseaux sociaux

Il existe de nombreuses plate-formes de réseaux sociaux en fonctionnement à travers Internet. Chacune dispose de plusieurs fonctionnalités permettant de la distinguer des autres. En conséquence les utilisateurs associés sont plus ou moins pertinents pour la marque [59]. On distinguera :

1- Les blogs et les Wikis.
2- Les plate-formes dites de micro-Blogging, comme Twitter.
3- Les réseaux sociaux publics, comme Facebook.
4- Les réseaux sociaux professionnels, comme LinkedIn ou Viadeo.
5- Les réseaux de marque-page social, dit Social Bookmarking, comme Babelio.
6- Les plate-formes d'étiquetage, dites de Tagging, comme Google PageRank.
7- Les plate-formes de contenu et de partage média, comme YouTube, Instagram et Flickr.
8- Les bon plans, dits Deal-of-the-day Platforms comme Groupon.
9- Les sites marchands comme eBay et Amazon.
10- Les plate-formes d'avis-consos, comme CNet.

5.3.1.2 La gestion de communauté sociale

Les médias sociaux sont riches d'émotions et d'opinions diffusés de plus en plus par rapport au nombre de personnes qui les partagent [60] [61]. Pour minimiser le risque de dérapage dans les réseaux sociaux, les entreprises et les marques ont besoin d'être au courant des sujets critiques relativement rapidement et ont besoin savoir comment réagir en situation de crise [62]. Afin de faire une gestion efficace de la communauté, il y a trois élément-clés à prendre en considération :

1- La marque doit publier son propre contenu afin de stimuler les discussions et les échanges entre les utilisateurs qui la suivent, dits Followers ou Fans.
2- La marque doit modérer les discussions pour s'assurer qu'elles ne partent pas dans des directions indésirables.
3- La marque doit surveiller ce qui se passe afin de prévoir et de prévenir tout méfait.

Les utilisateurs engagés dans des sujets critiques ont des attentes et des motivations différentes. Ainsi, s'occuper des sujets critiques permet d'éviter rapidement les incidents graves. Pour ce faire, il est important d'observer toujours les échanges au sein de la communauté et de réagir d'une manière adéquate. De plus, il est fortement recommandé de ne pas laisser les conversations sans surveillance, encore moins sous-estimer la puissance des influenceurs-clés [63]. Ce concept est illustré dans la Figure 53.

Figure 53 : Bases de la gestion des communautés

5.3.1.3 L'interaction avec les groupes cibles

Les médias sociaux offrent différentes manières d'interagir en collectivité avec les groupes cibles ou individuellement avec les consommateurs (des personnes à influence dans la plupart des cas). La capacité de dialoguer avec les utilisateurs connectés et d'échanger des messages est la plus importante, en termes d'influence sociale [64] [65].

Les principaux avantages pour une entreprise ou une marque que tous les médias sociaux possèdent en commun sont :

1- Le contact direct avec les consommateurs : c'est un levier de leur satisfaction.
2- L'authenticité des avis consommateur est un levier de la notation des moteurs de recherche.
3- L'interaction avec le consommateur permet d'obtenir rapidement et d'une manière abordable, le Feedback du consommateur et la connaissance du marché.

Les groupes cibles intéressants, avec des mesures spécifiques ou sur une plate-forme spécifique, sont stockés dans la base de données de l'entreprise. Toutes les données valides et disponibles sont extraites localement pour être retravaillées. En cas d'indisponibilité, une décision ultérieure doit être faite selon l'urgence de la situation, sur la nature et l'ampleur de la réaction de l'entreprise, ainsi que sur le choix du réseau social dans lequel on doit s'exprimer [66].

Le choix de la plate-forme est très important pour l'activité dans les médias sociaux en tant que résultat des discussions abordant d'ores et déjà des sujets qui touchent la réputation sociale de l'entreprise (marques, produits, employés). Cela exige un suivi approfondi au préalable afin de trouver ces échanges et de les affecter à des plate-formes ou à des facilitateurs ayant un rôle de multiplicateur des échanges.

5.3.2 La surveillance des réseaux sociaux

Afin de suivre correctement les réseaux sociaux au niveau mondial, l'administrateur a besoin d'une mise-à-jour régulière par rapport à la présence de la marque dans ce canal. Il doit disposer d'outils de travail adaptés [67] en termes de surveillance d'activité et de traitement des données permettant d'identifier rapidement les situations critiques et de réagir d'une manière appropriée.

Dans ce paragraphe, on définit un modèle de surveillance des données des réseaux sociaux composé de trois étapes :

1- L'acquisition et le stockage des données.
2- La modélisation et l'identification des données à valeur ajoutée, indispensables pour le calcul final.
3- Le traitement des données localement ou sur Internet.

5.3.2.1 L'acquisition des données

Les entreprises et les marques doivent faire face à des données en expansion continue en termes de volume, vitesse, variété et variabilité. Toutes les données utiles ont besoin d'être exploitées et utilisées. Ces données peuvent être collectées à partir des différentes plate-formes du canal des média sociaux. Elles peuvent être également croisées avec d'autres données en provenance de l'écosystème de l'entreprise. Généralement, ces données sont collectées à partir des sources d'information suivantes :

1- Internet, permettant d'importer des données générées par les utilisateurs (texte, images, audio, vidéo, documents) ou originaires des sites e-commerce (historique transactionnel et tendances d'achat).
2- Les outils CRM et données commerciales, comme les profils consommateur, en termes de comportement et d'orientation.
3- Les données publiques (statistiques et administratives) mises à disposition par les différents organismes.

5.3.2.2 La modélisation des données

Cette étape de pré-traitement permet d'identifier les données à valeurs ajoutée indispensables pour calculer le résultat final, parmi toutes les données importées des différentes sources et plate-formes. Grâce à ces sources diverses, les entreprises peuvent procéder à l'acquisition de données hétérogènes de différents types. Ces données peuvent être structurées (données contractuelles ou recueillies volontairement des consommateurs par le biais des services électroniques) ou non-structurées (données présentes essentiellement dans les médias sociaux et profils consommateurs dans les outils CRM de l'entreprise).

Les données les plus personnalisées et les plus à jour sont les plus précieuses parmi toutes. Les entreprises doivent convaincre les consommateurs qu'ils peuvent obtenir de meilleurs produits et services en produisant un retour d'information. Dans ce cas précis, les entreprises seront bien informées des tendances, préférences et attentes des consommateurs et les marques seront capables d'améliorer la satisfaction de leur consommateurs. Dans le but d'extraire les données à valeur ajoutée nécessaires pour le calcul du résultat final, il est important d'utiliser un ou plusieurs opérateurs de modélisation.

On rappelle les principales catégories de modélisation des données :

1- Les techniques conceptuelles capables d'identifier le plus haut niveau de liaison entre les différentes entités (duplication, agrégation, jointures).
2- Les techniques générales utilisées plus fréquemment dans les entrepôts de données (réduction dimensionnelle, tables d'index, clés énumérables).
3- Les techniques hiérarchiques permettant d'organiser les données dans des structures en arborescence en représentant les informations avec des nœuds parent / enfant (agrégations des arborescences, listes d'adjacence, chemins énumérés).

Ces outils classiques de modélisation des données doivent être couplés à un algorithme efficace de pré-traitement afin d'obtenir de meilleurs performances. On va expliquer par la suite comment mettre en œuvre le pré-traitement par étude de cas et d'envoyer au serveur de traitement, les données indispensables pour le calcul du résultat final requis seulement.

5.3.2.3 Le traitement des données

De nos jours, des plate-formes de traitement intuitives sont à disposition des utilisateurs novices leurs permettant de lancer des programmes de calcul de longue durée à travers des réseaux de serveurs de calcul. De plus, il est désormais possible pour les entreprises d'augmenter les capacités de stockage à des coûts raisonnables. Les outils de gestion BigData basés sur Hadoop et les techniques NoSQL ont permis aux entreprises d'optimiser leurs investissements en utilisant les plate-formes disponibles sur le Cloud en tant que services [68].

Vu que les utilisateurs payent pour ce qu'ils consomment, les réseaux de serveurs évolutifs ont permis de réduire les coûts en regroupant les périodes de calcul. Il est possible de réduire encore davantage les coûts en utilisant des techniques de modélisation efficaces et en mettant en œuvre une chaîne de traitement des données optimisée. Cette optimisation doit également mettre en œuvre la compression de données, une technique traditionnelle et efficace de gestion des données permettant de préserver les ressources des serveurs et du réseau. Plusieurs algorithmes de compression sont disponibles (LZO, GZIP et TAR). Ces algorithmes peuvent être utilisés en entrée / sortie pour traiter de larges DataSets de données. Toutes les solutions de Hadoop Framework sont non seulement compatibles avec les techniques de compression mais elles sont encore plus efficaces. En d'autres termes, la compression des données permet de réduire le nombre de ressources utilisées et par conséquent les coûts de traitement sur le Cloud.

5.3.3 La surveillance par étude de cas

Les entreprises et les marques cherchent à enrichir leur connaissance des consommateurs et des comportements grâce à la segmentation par profil [69] [70]. Ceci est piloté par l'exploitation des données en interne (CRM, données transactionnelles) et en externe (visites du site Internet, réseaux sociaux) combinées ensemble. Les offres et les services des marques peuvent être alors disponibles selon les profils et les préférences des consommateurs. De plus, la segmentation des consommateurs pourrait être affinée afin de procéder à la prédiction de la future tendance du marché [71] [72]. Par conséquent, les produits de la marque subissent une adaptation selon les études visant les consommateurs.

Amazon, Google, eBay, Yahoo et d'autres géants du Web sont capables de nos jours d'avoir une vue détaillée des profils de leurs consommateurs en connaissant en détail la nature des produits qu'ils achètent, les magazines et les journaux qui les attirent et même les sites Internet où ils naviguent. Tous les services fournis aujourd'hui sur Internet gardent une trace détaillée de l'activité des utilisateurs sur les serveurs de leurs FAI, indépendamment des contraintes juridiques de chaque pays. Cette connaissance profonde des besoins consommateur aide les entreprises à adapter leurs offres, à réduire le taux d'attrition et améliorer par conséquent leur chiffre d'affaires. Il existe donc trois catégories de profils :

1- Les profils professionnels basés sur l'offre et la demande dans les agences et les sites Internet spécialisés, les fournisseurs de cartes de visite, les mails et notifications reçu des médias sociaux professionnels (LinkedIn, Viadéo).
2- Les profils personnels (informations privées et données juridiques) tels que l'identité, les mails personnels, les commentaires et les notifications dans les réseaux sociaux (Facebook, Twitter, blogs).
3- Les profils commerciaux tels que les offres et les programmes de fidélité, les ventes privées ou les Newsletters que les consommateurs reçoivent de la marque.

Notre chaîne de traitement implémente différents profils de la catégorie des profils personnels représentés dans le Tableau 7 et basés sur l'activité d'un fabricant d'appareils électroménagers multinational.

Groupe ciblé / Objectif	**Consommateurs ordinaires**	**Leaders d'opinion**	**ONGs**	**Associations industrielles / organisations**
Employeur attractif	0	++	0	0
Leader d'innovation	0	++	0	+
Entreprise à actions responsables	++	++	++	++
Présence sur Internet et dialogue en ligne	0	++	++	++
Centre de compétences pour efficacité énergétique	0	++	++	++
++ très pertinent + pertinent 0 non important				

Tableau 7 : Profils personnels dans les données sociales en électroménager

D'autres profils commerciaux sont également montrés dans le Tableau 8.

Groupe ciblé / Objectif	Compte Clé	Petit et moyen compte	Prospect	En attrition
Fidèle à la marque	++	++	0	0
Pilote pour des nouveautés	+	+	0	0
Solde de crédit	++	+	0	++
Activité en litiges	++	++	0	++
Activité après-vente	++	+	0	++
Adhérent à la Newsletter	+	++	++	++
Présence sociale	++	++	+	++
Carte de fidélité	+	++	0	0
Ancienneté > 3 ans	++	++	0	++
Ancienneté > 5 ans	++	++	0	++
++ très pertinent + pertinent 0 non important				

Tableau 8 : Profils commerciaux dans les données sociales en électroménager

Les données non-relationnelles (réseaux sociaux, activité sur le site Internet, Newsletters) sont souvent liées aux requêtes spécifiques relatives aux applications, contrairement à la modélisation relationnelle. Les techniques de modélisation des données seront alors entraînées par des patrons d'accès spécifiques à l'application :

1- La compréhension profonde des structures de données est indispensable de façon à ce que la conception se concentre sur les questions qui correspondent aux données. De ce fait, on doit commencer par au moins un objectif métier en tête qui nous servira comme point de départ.
2- L'affectation des données aux rôles correspondants, vu que tous les rôles ne sont pas forcément adaptés aux mêmes types de données.
3- La fréquence et la quantité des données en changement continu.
4- La connaissance de la personnalisation des formules nécessaire, puisque les médias sociaux ne se soumettent à aucune norme.

Durant la phase de pré-traitement, ces profils seront considérés comme des instances indépendantes de raisonnement par étude de cas [73] et seront croisés avec les autres données dans le but de faire une première segmentation selon le critère de profil (valeurs de données). Un tel processus ne prend pas beaucoup de temps parce qu'au lieu d'analyser toutes les données disponibles, on prend seulement le résultat prédéfini de l'identification par étude de cas. Ces résultats seront alors agrégés et envoyés au serveur de calcul pour le traitement final.

Dans ce processus d'identification, on peut définir un certain nombre de variables. Ces variables sont alors groupées ensemble pour décrire les différentes segmentations de consommateurs. Certaines variables peuvent faire partie de plusieurs segmentations. Dans tous les cas, les données résultantes doivent s'étendre au-delà des informations évidentes.

5.3.4 Conclusion

Le monde d'aujourd'hui n'est qu'un réseau d'information gigantesque utilisant les médias sociaux à travers les Smartphones et les Tablettes. Les entreprises sont en discussion permanente avec les consommateurs pour toute décision rapide et efficace à prendre, vu que les consommateurs s'attendent à avoir des produits de bonne qualité, des services sur mesure et un retour d'information rapide [74]. De ce fait, il est important de les écouter et de comprendre leurs tendances et attentes afin d'améliorer les produits de la marque. De nos jours, le développement d'une marque est soumis à la qualité des informations. Le retour d'information à jour du consommateur aide les entreprises à anticiper les besoins du marché. Les meilleures campagnes Marketing sont celles capables de proposer les bons produits aux bons consommateurs, au bon moment. De plus, la réputation de la marque est soumise à son image sociale [75]. Les analyses sociales permettent de contrôler et d'améliorer la perception de la marque par la communauté sociale.

Plusieurs étapes sont nécessaires afin de déployer une présence sociale réussie de la marque :

1- Une correspondance entre l'identité des utilisateurs dans les réseaux sociaux et la base de données CRM de l'entreprise. L'élément d'identification principal peut être l'adresse mail utilisée dans les réseaux sociaux ou toute autre information privée fournie par les jeux sociaux, les échanges de services gratuits ou remises sur certains produits (fonctionnement soumis aux contraintes juridiques pour certains pays).
2- Une surveillance permanente des médias sociaux en utilisant des APIs spécifiques et en capturant les données actuelles et historiques, en termes de messages et discussions. Chaque message correspond à un sentiment spécifique [76] (positif, neutre, négatif) pouvant être utilisé ultérieurement dans l'interprétation des données.
3- Une utilisation d'outils performants de traitement des données permettant de détecter rapidement les situations critiques et de réagir efficacement en prenant les bonnes décisions.

5.4 L'apprentissage automatique pour les nouvelles situations

La puissance du modèle proposé se traduit en agrégeant des données hétérogènes dans une structure unique (par exemple, en cas d'étude de température d'un côté et des Tweets de l'autre, pour un département donné, à une date donnée). Cette puissance augmente en y incluant des fonctionnalités d'apprentissage automatique pour les nouvelles situations non prédéfinies. Cet apprentissage automatique est plus efficace, en utilisant une architecture à base de réseaux de neurones.

5.4.1 L'architecture du modèle à définir

Cette architecture est définie par 3 paramètres :

1- Les motifs d'interconnexion entre les différents profils de données et les conditions d'indicateurs KPI.
2- Le processus d'apprentissage pour mettre à jour les modèles de prédicats, en utilisant les statistiques approximatives [77] [78], permettant d'enrichir le système expert (système d'apprentissage en ligne).
3- La fonction d'activation, permettant d'initialiser le traitement des données à la sortie du moteur d'inférence.

5.4.2 Conclusion

L'apprentissage soulève un certain nombre de problématiques de recherche. Les questions qui se posent sont avant tout de savoir quelles sont les connaissances qui doivent être apprises et comment les apprendre. La plupart des recherches portent sur l'apprentissage des cas passés résolus, des méthodes d'indexation et de l'organisation de la base de cas, mais plus rares sont les recherches qui s'intéressent à l'apprentissage de connaissances implicites telles que les connaissances de similarité ou d'adaptation. De nos jours, des algorithmes disponibles sur Internet [79] peuvent être utilisés pour obtenir un apprentissage plus performant.

5.5 Conclusion du chapitre

Au cours des dernières années un nombre important de sources de données est devenu disponible et à la portée de tous. Les modèles de base à créer en utilisant les nombreux indicateurs uniques disponibles sur Internet peuvent être appliqués à beaucoup de cas d'emploi dans les domaines du Marketing, de la vente et de l'après-vente. Des interactions consommateur à valeur ajoutée utilisées pendant le traitement sur le Cloud peuvent être également être réexportées, dans le but d'améliorer la productivité des équipes. De simples opérateurs de modélisation connectables fournissent une vue détaillée pour mieux comprendre les besoins du consommateur. Un des objectifs principaux de l'utilisation des modèles de prédiction dans le traitement sur le Cloud est d'améliorer les performances du traitement avec des coûts moins important. Le modèle proposé dans ce chapitre permet d'étendre le traitement des données, en définissant des prédicats dans la base de données, basés sur un ensemble de règles prédéfinis.

Chapitre 6. Les résultats expérimentaux

6.1 Introduction au chapitre

Le chapitre précédent a défini un modèle dynamique pouvant être appliqué dans le traitement des données à base de cas. On a expliqué les composantes de ce modèle se basant sur l'acquisition et le stockage des données, l'identification des structures et des types de données et l'établissement des liaisons entre les entités. Ensuite, on a défini les procédures d'utilisation des systèmes experts, permettant d'améliorer les performances de traitement des données. Enfin, on a appliqué ce concept dans le domaine du traitement des données en entreprise, ainsi que dans le domaine des réseaux sociaux.

Dans ce chapitre, on va passer à la pratique afin de déterminer des cas d'emploi mettant en avant la modélisation intégratrice en général et le pré-traitement par étude de cas en particulier.

6.1.1 L'approche de la modélisation intégratrice

La modélisation intégratrice est aujourd'hui simple d'accès. Un utilisateur peut construire et appliquer son modèle de traitement à bases d'opérateurs de modélisation basiques ou à base de cas. L'approche réclame dans tous les cas des données de travail. Par exemple dans un contexte e-commerce, on dispose :

1- Des informations sur des caractéristiques client.
2- Des transactions réalisées entre comptes ou issues de capteurs de machines, collectées et stockées mais rarement analysées sous l'angle prédictif.

Le rôle d'un modèle de traitement à base de cas est de permettre non seulement de comprendre un phénomène mais surtout de retenir les données pertinentes noyées dans une masse d'information. Les dernières nouveautés en matière de moteurs de traitement accélèrent cette démarche en traitant des volumes importants de données en peu de temps. Enfin, il est bien entendu possible de traiter des données structurées et non-structurées à la fois, dans la même chaîne de traitement.

6.2 Les perspectives de la modélisation intégratrice

Dans ce paragraphe on va décrire plusieurs expériences mettant en œuvre la modélisation intégratrice dans un contexte BigData.

6.2.1 Les données Twitter

On a configuré le module d'acquisition de données de Twitter dans le but d'importer toutes les 10 secondes tous les Tweets publics correspondant aux mots-clés « crise » et « tristesse », l'objectif étant de trouver la corrélation entre ces deux mots-clés. On a donc importé 1000000 fichiers XML de données non-relationnelles selon le modèle orienté document. Ces fichiers sont composés d'une structure hétérogène complexe embarquant plusieurs dimensions (utilisateurs, rôles, coordonnées géographiques, Timestamps, Retweets) comme indiqué dans la Figure 54.

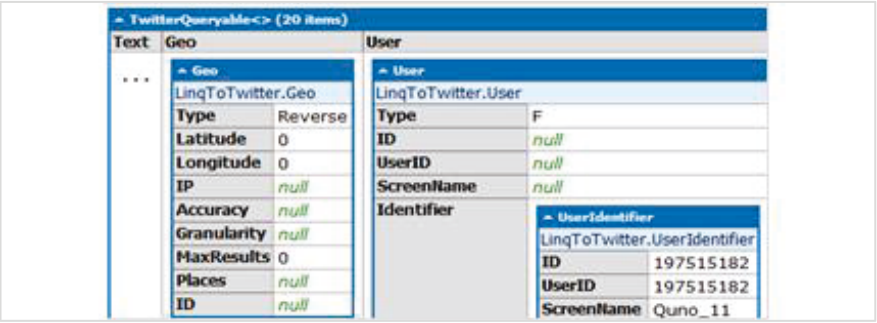

Figure 54 : Structure d'un Tweet

Dans le but d'améliorer les performances de traitement, on a implémenté en amont deux opérateurs de modélisation différents. Ainsi, les données en entrée du moteur de calcul Hadoop, pourront subir une modélisation et seront synthétisées d'une manière significative. Cette tâche consiste à utiliser la technique de modélisation par documents imbriqués, permettant de convertir un fichier XML à structure hiérarchique, en simple document texte plat, contenant seulement les données indispensables au calcul du résultat final de l'étude. On a donc sélectionné les colonnes suivantes :

1- Le champ UserID.
2- Le champ CreationDate.
3- Le champ Country.
4- Le champ Retweets.
5- Le champ PrivacyLevel.

On a utilisé par la suite un second opérateur de modélisation permettant d'agréger les données de l'ensemble des fichiers, selon une échelle de 1/1000. Finalement, on a envoyé le lot de 35 fichiers résultant du pré-traitement dans le moteur de calcul Hadoop, l'objectif étant de calculer la fréquence quotidienne d'apparition de ces mots-clés. Le calcul MapReduce est alors réalisé sur 3 niveaux indépendamment selon le diagramme d'activité de la Figure 55 :

1- D'abord sur les données initiales.
2- Ensuite un pré-traitement avec le premier opérateur de modélisation.
3- Finalement un pré-traitement avec deux opérateurs de modélisation.

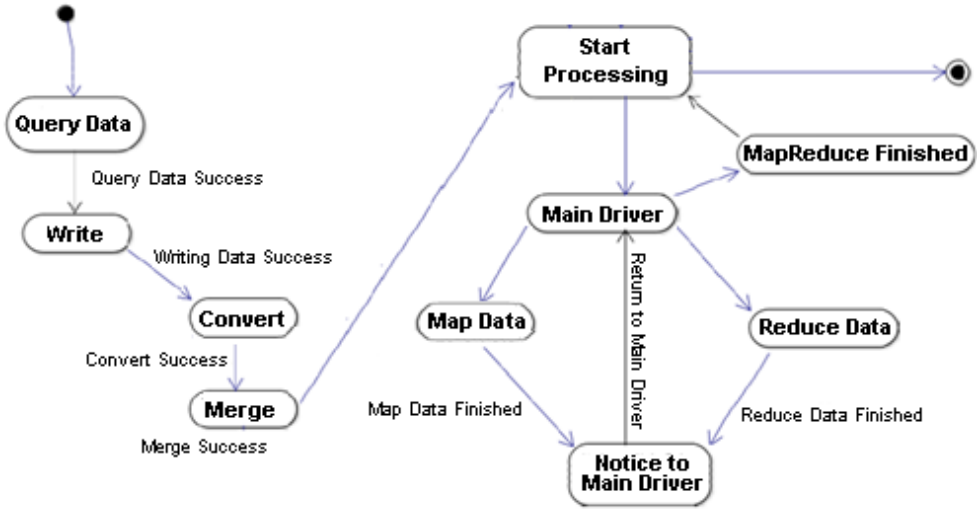

Figure 55 : Diagramme d'activité du traitement des données Twitter

Au troisième niveau, la taille des données a atteint 1,9 GO, après réduction de 72% à peu près par rapport à la taille initiale de 6,9 GO. Les résultats sont détaillés dans le Tableau 9.

Type de traitement	Taille des données	Durée de traitement
Données initiales	6,9 GO	> 20 h
+ opérateur documents imbriqués	2,4 GO	5 h 35 m
+ opérateur agrégation	1,9 GO	4 h 52 m

Tableau 9 : Utilisation des opérateurs de modélisation sur les données Twitter

Les résultats de calcul montrent que l'utilisation des opérateurs de modélisation a amélioré les performances de calcul de plus de 74%. Il est possible également d'étendre ce modèle en ajoutant d'autres opérateurs dans la chaîne de traitement.

6.2.2 BigData Workbench

L'expérience précédente met en avant l'impact d'utilisation des opérateurs de modélisation sur les données non-relationnelles. Afin d'implémenter une nouvelle abstraction basée sur une architecture MDA et dans l'optique d'un passage à l'échelle, un AGL a été conçu en mode agile permettant aux utilisateurs non développeurs de réaliser des chaînes de traitement des données non-relationnelles avec des composants de type glisser-déplacer. Ce logiciel, illustré dans la Figure 56, contient des fonctions pour :

1- Ajouter un ou plusieurs composants parmi les sources de données à disposition (fichiers de données, réseaux sociaux, Web Services).
2- Faire une analyse prédéfinie sur un échantillon de données, pour définir automatiquement la structure des fichiers / messages.
3- Utiliser un ou plusieurs opérateurs de modélisation des données non-relationnelle à disposition dans l'outil, en connectant les composants ensemble.
4- Sélectionner un ou plusieurs composants de moteurs de calcul, en mode local ou sur le Cloud.

On a la conviction qu'une telle conception logicielle aidera les utilisateurs à réduire la durée et les coûts de traitement :

1- En laissant à l'utilisateur l'initiative pour concevoir sa chaîne de traitement.
2- En appliquant une décentralisation du traitement sur un ou plusieurs moteurs de calcul.
3- En réduisant le volume des données à traiter.

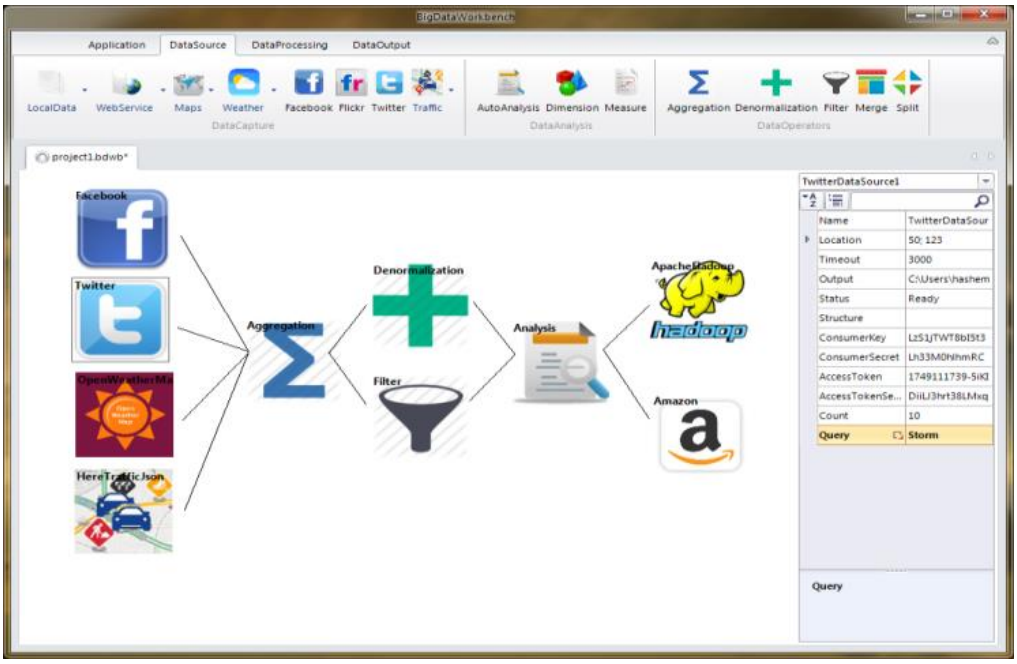

Figure 56 : BigData Workbench

6.2.2.1 Le pré-traitement avec BigData Workbench

Dans l'exemple de la Figure 56 on procède à l'acquisition des données depuis de multiples sources disponibles sur Internet :

1- Les données sociales de Facebook et Twitter.
2- Les données météo de l'API Weather.
3- Les données de trafic de l'API Here.

Comme dans l'expérience précédente, toutes les données capturées correspondent à un mot clé. Cette fois-ci, il s'agit du mot-clé « inondation ».

6.2.2.2 Les scénarii expérimentaux

L'objectif de l'expérience est de mesurer la fréquence d'apparition du mot clé cité. Cette étape correspond à traiter les données capturées selon trois configurations différentes :

1- Le scénario 1 consiste à traiter les données dans le serveur Hadoop sans aucun travail de modélisation. Dans cette étape, 25 GO de données sont envoyés au serveur de calcul.
2- Ensuite le scénario 2 consiste à utiliser l'opérateur d'agrégation sur les différents fichiers et avant de traiter les données sur le serveur de calcul. L'objectif est donc d'unifier le format de données indépendamment de la source lors du traitement afin d'éviter une série de traitements, dont chacun dépend d'un format donné différent.

3- Finalement le scénario 3 consiste à lancer le traitement avec deux autres opérateurs de modélisation, le filtre permettant d'identifier les données clés ayant un lien direct avec le résultat final et la dé-normalisation permettant de dupliquer ces données pour de meilleures performances de traitement.

Dans tous les scénarii, on utilise deux types de serveurs de calcul :

1- Un serveur Hadoop sur un réseau local (1 TO HDD - 2,5 GHZ CPU - 8 GO RAM - 10 Gbit/s).
2- Un serveur de calcul Amazon avec la même configuration matérielle.

6.2.2.3 Les résultats de traitement

On note les résultats de calcul de chacun des scenarii dans le Tableau 10.

	Data pre-processing	**Local Apache Hadoop server**	**Amazon Cloud server**
Scenario 1		>24h	>24h
Scenario 2	+23m	11h17m	10h42m
Scenario 3	+39m	5h21m	4h54m

Tableau 10 : Récapitulatif du traitement avec BigData Workbench

En se basant sur les résultats affichés, on trace le graphe de la Figure 57.

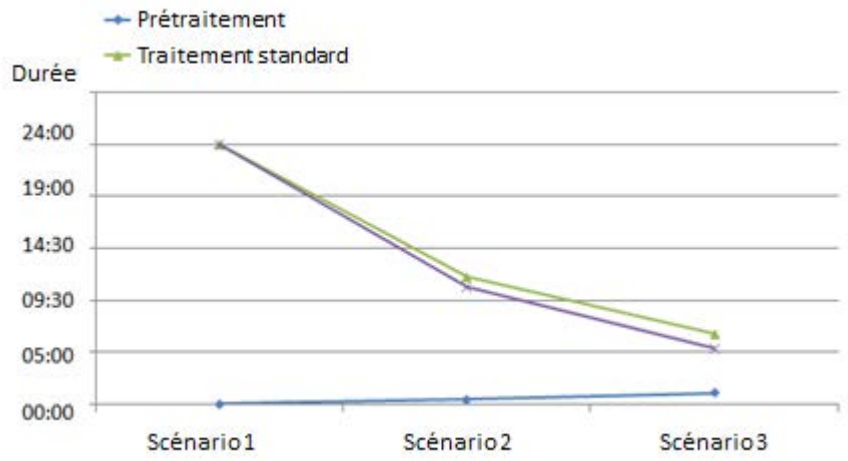

Figure 57 : Graphe récapitulatif du traitement avec BigData Workbench

6.2.3 Conclusion

En utilisant les opérateurs de modélisation on constate un traitement des données aboutissant à un résultat bien plus rapide. Dans l'expérience précédente, le temps consacré pour le pré-traitement avec les opérateurs de modélisation n'a pas dépassé une heure. Près de cinq heures au total était suffisantes pour avoir le résultat final, contrairement à un traitement standard sans opérateurs de modélisation. On constate également des résultats similaires entre le calcul en local et sur le Cloud, mis à part le temps de chargement nécessaire à transférer les données.

6.3 Le pré-traitement par étude de cas

Les cas d'emploi discutés par la suite ont pour but de montrer l'efficacité de l'utilisation des modèles de prédicats dans un traitement BigData.

6.3.1 Cas d'emploi 1 : L'évaluation du profil revendeur

Ce concept est appliqué sur les factures générales et spécifiques, promotions, remises de performances et gestes commerciaux. L'évaluation objective du revendeur d'une marque est faite à travers la connaissance de l'inspecteur commercial et son expérience avec le revendeur. Les directives de profil revendeur influencent différentes situations au sein de l'entreprise. Plusieurs fonctions prennent part au processus et sont affectées par le résultat et les décisions. Le but est donc de :

1- Recueillir des informations revendeur, permettant d'évaluer son profil correctement.
2- S'assurer de l'objectivité de l'étude, afin qu'un revendeur ne puisse pas profiter d'un traitement de faveur et qu'il soit évalué selon ses performances.

D'un point de vue technique, la force de vente (en industrie de fabrication d'appareils électroménagers par exemple), qui visite les revendeurs au quotidien, utilise des terminaux mobiles pour récolter les informations permettant de décrire le profil du revendeur comme suit :

1- Les critères décrivant le magasin en termes de surface, présentation des appareils, services à domicile, nombre de cuisines exposées, nombre de vendeurs, service après-vente.
2- Les critères décrivant le site Internet en termes de bannières, présence des fenêtres publicitaires, dits Popups, moyens et conditions de paiement, qualité et résolution des images, nombre de cliques pour accéder à la page de commande.
3- Les aspects décrivant la santé financière du revendeur en termes de chiffre d'affaire, niveau de risque de crédit, marge grossiste, sous-réseaux.

Dans une telle étude on définit des modèles de prédicats spécifiques aux modèles de revendeurs :

1- Les magasins électroménagistes indépendants.
2- Les chaînes régionales / nationales / internationales.
3- Les sites Internet, dits Pure-players.
4- Les commerçants multiple / hybride.
5- Les grossistes.
6- Les cuisinistes indépendants.
7- Les centrales d'achat.

En se basant sur ces modèles, on définit les règles de conditions permettant de réaliser une évaluation objective du revendeur :

1- La qualité de présentation des appareils.
2- La qualité du service après-vente.
3- La qualité de l'accueil téléphonique, dit Hotline.
4- La formation.

Chacune de ces conditions permettra de calculer en pourcentage une partie de la remise totale à accorder au revendeur. Les résultats seront agrégés ensemble puis subiront une pondération dans le but de calculer la remise globale dans les conditions commerciales. On se base alors sur les éléments suivants pour estimer la pondération:

1- La remise correspondant à l'efficacité du processus.
2- La remise correspondant à la qualité du magasin.
3- La remise correspondant au site Internet.
4- La remise grossiste.

Une fois toutes les informations rassemblées par la force de vente (processus à faire régulièrement), toutes les données stockées dans le terminal mobile seront transférées automatiquement au serveur central de l'entreprise via un mécanisme de synchronisation sur Internet. Alors un automate se lance (moteur d'inférence) et se met à calculer la remise globale correspondant à chacun des revendeurs en se basant sur les indicateurs KPI détectés dans les données. A la fin du traitement, les résultats seront transmis au serveur de facturation afin de prendre en considération les remises calculées. De nouvelles factures calculées en fonction des profils des revendeurs seront alors créées. A la fin du processus, ces factures seront mises à disposition en téléchargement sur le site Internet de la marque ou seront envoyées aux revendeurs (par courrier ou par mail). Cette démarche est illustrée dans la Figure 58.

Figure 58 : Evaluation du profil revendeur avec un pré-traitement par étude de cas

On a implémenté le processus d'évaluation des revendeurs en utilisant les données d'une entreprise de fabrication d'appareils électroménagers (données anonymes) dont le détail est affiché dans le Tableau 11.

Elément	Instances
Nombre de critères	188
Nombre de modèles de prédicats	10
Nombre de conditions	56
Nombre de règles de pondération	33
Total revendeurs	648
Total enregistrements de données à traiter	13043

Tableau 11 : Processus d'évaluation des revendeurs avec un pré-traitement par étude de cas

Afin de réaliser l'expérience on a mis en place un prototype en mode agile permettant de lancer la simulation avec les données de test. Cet outil est montré dans la Figure 59 et la Figure 60.

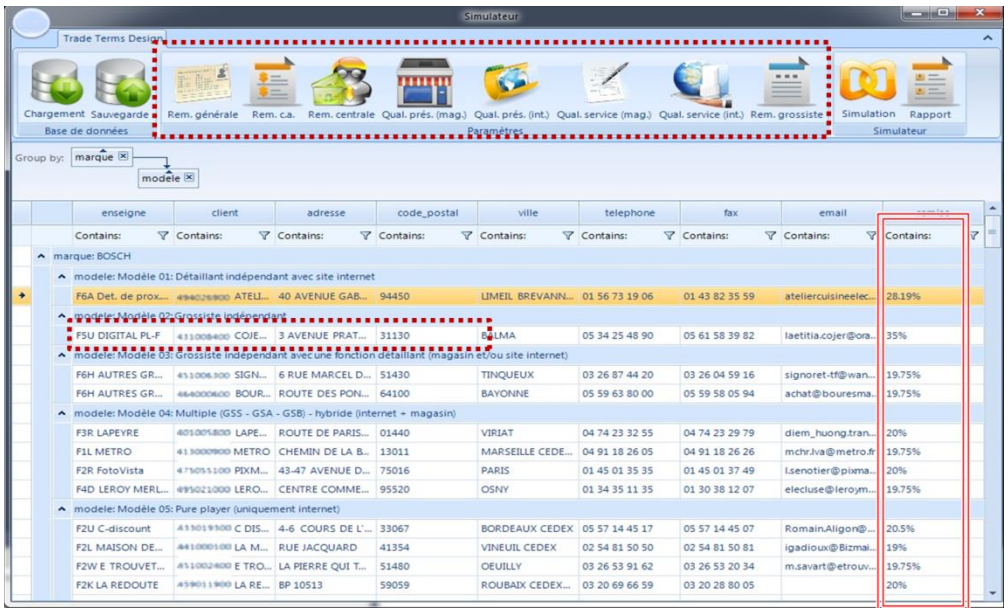

Figure 59 : Simulateur de traitement à base de cas

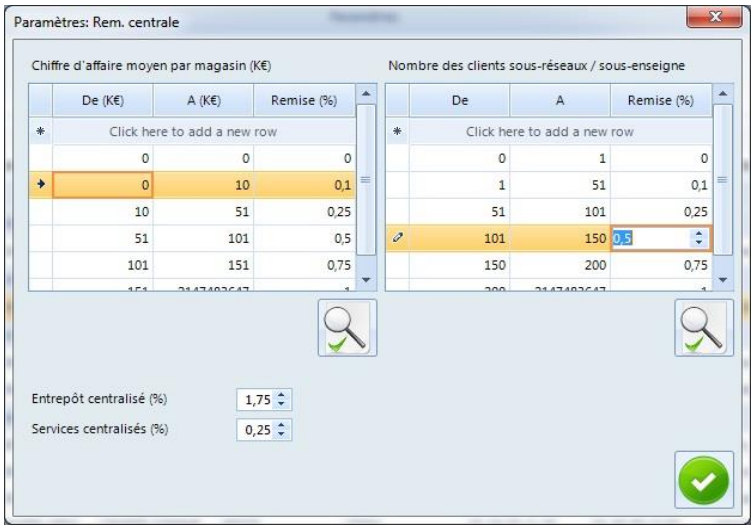

Figure 60 : Paramétrage du simulateur de traitement à base de cas

A la fin du traitement, les résultats de calcul du niveau de performance se présentent comme indiqué dans le Tableau 12.

Etape de traitement	Durée
Traitement de la pré-vérification	0,507 s
Traitement correspondant à la première étape	37,8 s
Traitement correspondant à la deuxième étape	22,275 s
Total	60,582 s

Tableau 12 : Résultats de calcul du pré-traitement par étude de cas

Dans le but de faire une comparaison, on a mis en place un script de traitement MapReduce sur un serveur de calcul Hadoop (dernière version stable disponible) possédant le même bagage matériel. Le résultat de ce traitement est affiché dans le Tableau 13.

Etape de traitement	Durée
13043 enregistrements de données en une fois	81,3 s

Tableau 13 : Résultats de calcul du traitement MapReduce

Ces résultats dépendent aussi de la complexité des données d'échantillonnage et les performances matérielles du serveur de calcul. Dans tous les cas, le temps de traitement des données en une fois est nettement plus grand que celui après un pré-traitement des données.

6.3.2 Cas d'emploi 2 : Les changements dans le trafic routier

Ce paragraphe considère la problématique du recalcul d'un trajet pour un véhicule suite à des incidents imprévus sur le parcours. La décision est alors basée sur des éléments factuels :

1- Les données météo.
2- Les points de contrôle de police.
3- Les radars, feux, ralentisseurs et passages Stop.
4- Les obstacles et routes barrées.
5- Les zones de travaux.
6- Les accidents de circulation.
7- Le nombre d'automobilistes sur le même trajet

La décision est basée aussi sur tout autre élément en relation, de près ou de loin, avec les éléments suivants :

1- La carte de données de trafic routier qui fournit la moyenne du relevé de trafic sur la route empruntée.
2- Les indicateurs de temps de parcours, qui fournissent les données correspondant entre le point de départ et le point d'arrivée.
3- Les données spécifiques de la circulation, permettant de connaître les derniers arrangements de transport spécial, à travers le territoire, en termes d'accidents de circulation.

Le concept à implémenter consiste à faire une acquisition des données de trafic routier en utilisant une antenne GPS disponible dans le terminal mobile du conducteur [80]. Ces données seront transmises au serveur central de calcul afin d'être combinées avec d'autres informations de trafic centralisées [81], comme les données météo, les informations générales de circulation ou autres.

Le moteur d'inférence fera la distinction entre les données indispensables pour le calcul du résultat et les autres données non indispensables [82]. A la fin du traitement, le résultat est retransmis via la connexion Internet aux usagers de la route pour les avertir en cas d'ajustement nécessaire de la carte de route. Cette même information transmise également au serveur de navigation, ce qui permettra de leur proposer un ou plusieurs trajets alternatifs selon leur position actuelle sur la route. Cette démarche est illustrée dans la Figure 61.

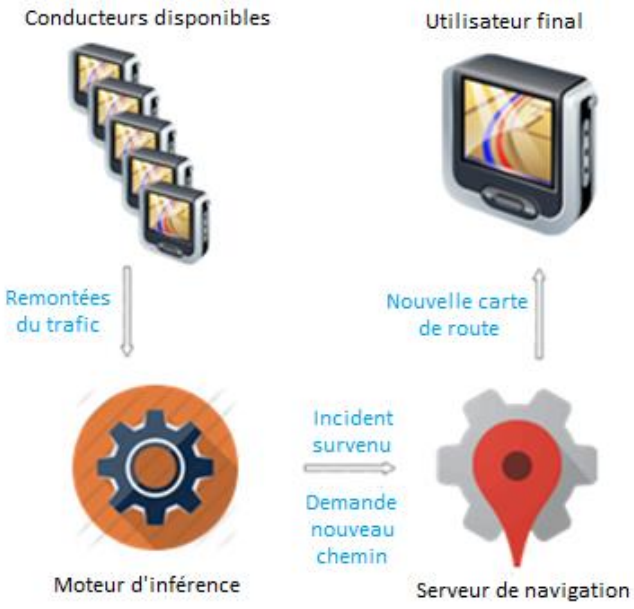

Figure 61 : Adaptation du trajet selon le trafic routier

6.3.3 Cas d'emploi 3 : La détection d'un taux d'attrition élevé

Les taux d'attrition s'élèvent lorsque les consommateurs décident de changer d'une marque à une autre pour obtenir de meilleurs services ou de prix moins chers. Cela se produit en raison de la concurrence et de la saturation du marché par les marques alternatives ayant tendance à attirer les consommateurs. Les entreprises stockent dans leurs bases de données des informations consommateur telles que la satisfaction ou la fidélité permettant d'effectuer une prédiction du comportement du consommateur vis-à-vis de la marque.

Le défi est la détection des consommateurs à faible potentiel, avant le basculement à une autre marque. Les marques devraient mieux se concentrer sur cette tâche que de chercher à gagner de nouveaux consommateurs. Les consommateurs doivent alors subir une segmentation selon des modèles de prédicats prédéfinis :

1- Le consommateur fidèle.
2- Le consommateur à risque.
3- Le consommateur perdu.

Cette segmentation permet de mieux comprendre les intentions des consommateurs. Cela coûte plus cher de chercher à gagner à nouveau un consommateur venant d'une autre marque, que de garder un niveau de satisfaction élevé d'un consommateur déjà existant. La détection du taux d'attrition dans ce cas permet d'identifier les consommateurs à risque avant qu'ils prennent l'initiative de changer de marque et ce grâce aux opérateurs de modélisation.

Les profils consommateurs permettent de mieux connaître leurs tendances et leur comportement. Les opérateurs de modélisation des données permettent d'analyser les profils consommateurs et les variables correspondantes [83], y compris les données en rapport avec la démographie, la saisonnalité, l'utilisation et les offres de la compétition.

Les indicateurs d'un taux d'attrition élevé comprennent en général :

1- Le refus d'utilisation des services et des produits proposés par la marque.
2- Le nombre élevé des appels avec le service après-vente.
3- Les litiges entre le consommateur et la marque.
4- Les retards de paiement.

L'hypothèse de la mise en œuvre d'un processus d'inférence conduirait à définir un certain nombre de variables qui, groupées ensembles, permettront de décrire les différentes catégories consommateurs définies auparavant. Certaines variables peuvent être utilisées dans plusieurs catégories. Dans tous les cas, le résultat de traitement des données doit s'étendre au-delà des informations flagrantes. Le moteur d'inférence serait alors en mesure de fournir de nouvelles informations permettant d'effectuer la prédiction recherchée (par exemple, la prédiction du comportement d'un consommateur n'ayant souscrit à aucun service depuis plusieurs mois).

6.3.4 Conclusion

Ce type d'approche donne la possibilité de pouvoir comprendre et anticiper les comportements (d'un client, d'une entreprise, d'un matériel) avant qu'ils ne surviennent afin de mettre en place des pistes d'amélioration ou des actions correctives. Les méthodes utilisées sont parfois anciennes et reposent en toute simplicité sur l'analyse des données réelles à disposition. Mais la contribution de cette approche se présente en appliquant ces méthodes dans un contexte de traitement volumineux à architecture distribuée.

Dans tous les domaines, les bénéfices métier sont nombreux :

1- Proposer la meilleure offre pour répondre au besoin d'un consommateur.
2- Minimiser les comportements déviant (risque, fraude, abus) avant qu'ils ne se propagent.
3- Planifier une intervention de maintenance sur un matériel, dont un composant présente des signes de faiblesse.

6.4 Conclusion du chapitre

Dans ce chapitre on a procédé à l'expérimentation de la modélisation intégratrice qu'on a conçue. On a tout d'abord construit un modèle de traitement des données Twitter avec deux opérateurs de modélisation : les documents imbriqués et l'agrégation. Ensuite, on a construit une chaîne complète de traitement avec un outil conçu pour cet usage. Cette deuxième expérience traitant des données hétérogènes et de sources différentes met en évidence à travers trois scenarii utilisés, l'impact des opérateurs de modélisation dans le cadre d'un pré-traitement des données et avant envoi au serveur de calcul. Finalement, on a procédé aux expérimentations relatives au pré-traitement par étude de cas. On a mis en lumière le traitement des données client en entreprise, avec le simulateur conçu pour cet usage. On a procédé à l'expérimentation de deux autres cas d'emploi concernant la gestion du trafic routier et la détection des taux d'attrition élevés.

Tous ces scénarii ont été réalisés afin d'illustrer l'impact des opérateurs de modélisation sur la chaîne de traitement des données BigData. On a mis en exergue notamment le raisonnement par étude de cas permettant d'obtenir un résultat préliminaire par simple comparaison de modèles renseignés dans une base de connaissance.

Partie 3. Conclusion et perspectives

Conclusion

En parallèle à l'étendue infinie de la volumétrie des données BigData échangées sur Internet, ainsi qu'à l'évolution exponentielle des services de traitement de ces données avec les outils de l'univers Hadoop, la modélisation intégratrice se positionne comme une étape de haute valeur ajoutée permettant d'atteindre le résultat recherché avec moins d'efforts et de ressources.

Dans cette étude, on a adressé cette problématique en proposant un concept général qui s'adapte à tout besoin et reposant sur une boîte à outils offrant plusieurs opérateurs de modélisation des données. On a proposé dans ce travail également une technique de pré-traitement avec un raisonnement par étude de cas. Globalement, cette étude a été conduite comme suit :

- **Le traitement des données BigData et l'évolution des systèmes de gestion des bases de données non relationnels** : On a présenté le modèle de données NoSQL et sa dérivée NewSQL, en termes d'architecture, d'avantages et de limitations. On a évoqué par la suite une analyse de l'efficacité des moteurs de traitement existant de nos jours. Ensuite, on a défini les modèles de données d'une base non-relationnels, notamment le stockage clé-valeur, le modèle orienté colonne, le modèle orienté document et le graphe. On a parlé également des bases de données multi-modèle et de leur usage. Enfin, on a discuté de l'activité principale des systèmes distribués et des difficultés de leur mise en œuvre.
- **Le Framework MapReduce et ses principales caractéristiques :** On a présenté les différentes techniques de traitement et patrons de conception, tels que le tri, les jointures, l'indexation, le classement et la conversion. On a présenté également les algorithmes de traitement des graphes et de traitement du texte, ainsi que les principaux projets dans l'univers Hadoop tels que YARN, Apache Storm et Apache Spark.
- **La modélisation des données BigData :** On a présenté les recherches précédentes portant sur l'approche de la modélisation intégratrice et on a détaillé les 3 familles des techniques de modélisation, la modélisation conceptuelle, la modélisation générale et la modélisation hiérarchique. Par la suite, on a élaboré notre concept de traitement à base d'opérateurs de modélisation et son impact sur les performances de la chaîne de traitement BigData.
- **Les algorithmes de modélisation avec MapReduce :** On a déterminé les algorithmes correspondant aux principaux opérateurs de modélisation, tels que le filtre, le découpage, la transformation ou la fusion, ainsi que les patrons basiques et non-basiques de MapReduce et les patrons relationnels. On a défini les algorithmes d'agrégation, d'assemblage, le tri, les tâches distribuées et les algorithmes de traitement des graphes. On a défini également les algorithmes de sélection, d'intersection, de projection, d'union et des jointures. Enfin, on a présenté l'API Trident d'Apache Storm et les potentiels de l'apprentissage automatique avec MapReduce.

- **Le pré-traitement selon un raisonnement par étude de cas :** On a présenté les systèmes experts et le potentiel d'un rapprochement avec les systèmes de gestion des bases de données. On a élaboré le concept de moteur d'inférence avec une base de connaissance de prédicats. Ce modèle étant un moyen efficace pour lancer un pré-traitement des données BigData en se basant sur des cas similaires et permettant par conséquent d'arriver rapidement à une indication sur le résultat final, avec un niveau de tolérance raisonnable. Finalement, on a illustré l'application de ce modèle dans le contexte d'une entreprise de fabrication d'appareils électroménagers et dans le contexte des réseaux sociaux.
- **La boîte à outils de modélisation BigData :** Le rapprochement entre les systèmes experts et les systèmes de gestion des bases données relationnels ou non-relationnels a permis d'obtenir une boîte à outils personnalisables, permettant de créer une chaîne de traitement basée sur des opérateurs de modélisation à la carte et profitant des performances de calcul de Hadoop MapReduce.

En conclusion, on a proposé un modèle intuitif clé-en-main ne nécessitant pas une connaissance technique approfondie dans un domaine technologique en expansion continue et ayant un impact positif sur les performances de la chaîne de traitement. On peut désormais parler d'un traitement BigData utilisant les règles du raisonnement par étude de cas, à l'échelle des réseaux distribuées et garantissant un traitement décentralisé, séquentiel, isolé du développeur et évolutif selon le besoin en vigueur.

Les perspectives

Ce travail a été effectué pour deux motivations principales: (1) proposer un concept de modélisation intégratrice à base d'opérateurs adaptables à toute chaîne de traitement BigData. (2) implémenter ce concept pour des utilisateurs n'ayant pas une connaissance technique approfondie dans ce domaine et possédant des besoins précis.

Cette approche repose sur trois étapes. D'abord, l'acquisition des données diverses et variées, de sources multiples, de tailles et de formats différents, à travers des connecteurs assurant une conversion des flux en fichiers à stocker, selon le modèle de base de données utilisée. Ensuite, le formatage des données via des opérateurs de modélisation sélectionnés par l'utilisateur, selon une configuration précise et adéquate avec son besoin. De ce fait, on peut utiliser de simples opérateurs tels que la fusion, le tri ou la dé-normalisation. On peut également mettre en place des algorithmes plus sophistiqués se basant sur un raisonnement par étude de cas. Enfin, mettre en œuvre le traitement des données présélectionnées dans le moteur de calcul Hadoop, via le processus classique de MapReduce.

De ce fait, il est toujours possible d'optimiser ces concepts par :

1- L'enrichissement de la boîte à outils de modélisation par inclusion d'autres algorithmes plus sophistiqués, tels que des Workflows ou des algorithmes de traitement de son et d'image.
2- La mise-à-niveau de la base de prédicats en y incluant des capacités d'apprentissage automatique, en utilisant les algorithmes d'intelligence artificielle.

Une autre direction de développement de ce travail serait d'intégrer le concept de la modélisation intégratrice et du raisonnement par étude de cas dans le Framework Hadoop MapReduce. Cette approche permettrait à l'utilisateur de gérer moins d'outils et par conséquent de faciliter la manipulation des données dans un seul et même outil, regroupant à la fois l'acquisition, la modélisation et le traitement.

Bibliographie

[1] S. Perera et T. Guanarathne, Hadoop MapReduce Cookbook, Packt Publishing, 2013, Print ISBN 978-1-84951-728-7.

[2] G. Wang et J. Tang, The NoSQL Principles and Basic Application of Cassandra Model, International Conference of Computer Science & Service System (CSSS), 2012, Print ISBN 978-1-4673-0721-5.

[3] Y. Li et S. Manoharan, A performance comparison of SQL and NoSQL databases, IEEE Pacific Rim Conference of Communications, Computers and Signal Processing (PACRIM), 2013, ISSN 1555-5798.

[4] B. G. Tudorica et C. Bucur, A comparison between several NoSQL databases with comments and notes, International Conference of Networking in Education and Research (RoEduNet), 2011, Print ISBN 978-1-4577-1233-3.

[5] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes et R. Gruber, Bigtable: A Distributed Storage System for Structured Data, USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2006, Print ISBN 1-931971-47-1.

[6] M. N. Vora, Hadoop-HBase for large-scale data, International Conference of Computer Science and Network Technology (ICCSNT), 2011, Print ISBN 978-1-4577-1586-0.

[7] S. Ghemawat, H. Gobioff et S. K. Leung, The Google File System, ACM Symposium on Operating Systems Principles (SOSP), 2003, Print ISBN 1-58113-757-5.

[8] K. Kaur et R. Rani, Modeling and querying data in NoSQL databases, IEEE International Conference on BigData (BigData), 2013, INSPEC Accession Number 13999217.

[9] B. Yu, A. Cuzzocrea, D. Jeong et S. Maydebura, On Managing Very Large Sensor-Network Data Using Bigtable, IEEE International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012, Print ISBN 978-1-4673-1395-7.

[10] S. Lai, Graph-theory model based E-commerce website design and realize, International Conference on Computing and Networking Technology (ICCNT), 2012, Print ISBN 978-1-4673-1326-1.

[11] E. Anderson, C. Hoover, X. Li et J. Tucek, Efficient tracing and performance analysis for large distributed systems, IEEE Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2009, Print ISBN 978-1-4244-4927-9.

[12] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu et A. Wolman, Volley: Automated Data Placement for Geo-Distributed Cloud Services, USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2010, Print ISBN 978-931971-73-7.

[13] D. J. DeWitt et J. Gray, Parallel database systems: The future of high performance database systems, Communications of the ACM, 1992, DOI 10.1145/129888.129894.

[14] H. Karloff, S. Suri et S. Vassilvitskii, A model of computation for MapReduce, ACM-SIAM Symposium on Discrete Algorithms (SODA), 2010, Print ISBN 978-0-898716-98-6.

[15] B. Schroeder, E. Pinheiro et W. D. Weber, DRAM errors in the wild: A large-scale field study, International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), 2009, Print ISBN 978-1-60558-511-6.

[16] E. Pinheiro, W. D. Weber et L. A. Barroso, Failure trends in a large disk drive population, USENIX Conference on File and Storage Technologies (FAST), 2008, Print ISBN 978-1-931971-66-9.

[17] C. Olston, B. Reed, U. Srivastava, R. Kumar et A. Tomkins, Pig Latin: A not-so-foreign language for data processing, ACM International Conference on Management of Data (SIGMOD), 2008, Print ISBN 978-1-60558-102-6.

[18] J. Lin, A. Bahety, S. Konda et S. Mahindrakar, Lowlatency, high-throughput access to static global resources within the Hadoop framework, Technical Report HCIL-2009-01 University of Maryland, 2009.

[19] A. Moffat, W. Webber et J. Zobel, Load balancing for term distributed parallel retrieval, ACM Conference on Research and Development in Information Retrieval (SIGIR), 2006, Print ISBN 1-59593-369-7.

[20] R. Xu et D. W. II, Survey of clustering algorithms, IEEE Transactions on Neural Networks (TNN), 2005, DOI 10.1109/TNN.2005.845141.

[21] P. Hage et F. Harary, Island Networks: Communication, Kinship, and Classification Structures in Oceania, Cambridge University Press, 1996, Print ISBN 978-0-52103-321-3.

[22] L. Page, S. Brin, R. Motwani et T. Winograd, The PageRank citation ranking: Bringing order to the Web, Stanford Digital Library of Stanford University, 1999, Working Paper SIDL-WP-1999-0120.

[23] A. P. Dempster, N. M. Laird et D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society, 1977, JSTOR 2984875/MR0501537.

[24] F. Jelinek, Statistical methods for speech recognition, MIT Press, 1997, Print ISBN 0-262-10066-5.

[25] K. Seymore, A. Mccallum et R. Rosenfeld, Learning hidden Markov model structure for information extraction, International Workshop on Machine Learning for Information Extraction (AAAI), 1999, DOI10.1.1.320.6302.

[26] M. Stanke et S. Waack, Gene prediction with a hidden Markov model and a new intron submodel, Bioinformatics, 2003, DOI 10.1093/BTG1080.

[27] D. Cutting, J. Kupiec, K. Pedersen et P. Sibun, A practical part-of-speech tagger, International Conference on Applied Natural Language Processing (ANLC), 1992, DOI 10.3115/974499.974523.

[28] M. R. Hassan et B. Nath, Stock market forecasting using hidden Markov models: A new approach, International Conference on Intelligent Systems Design and Applications (ISDA), 2005, Print ISBN 0-7695-2286-06.

[29] D. R. H. Miller, T. Leek et R. M. Schwartz, A hidden Markov model information retrieval system, ACM Conference on Research and Development in Information Retrieval (SIGIR), 1999, Print ISBN 1-58113-096-1.

[30] S. Vogel, H. Ney et C. Tillmann, HMM-based word alignment in statistical translation, International Conference on Computational Linguistics (COLING), 1996, DOI 10.3115/993268.993313.

[31] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall et W. Vogels, Dynamo: Amazon's highly available key-value store, ACM Symposium on Operating Systems Principles (SOSP), 2007, DOI 10.1145/1323293.1294281.

[32] N. Alon, Y. Matias et M. Szegedy, The space complexity of approximating the frequency moments, ACM Symposium on Theory of Computing (STOC), 1996, Print ISBN 0-89791-785-5.

[33] A. Levenberg et M. Osborne, Stream-based randomised language models for SMT, International Conference on Empirical Methods in Natural Language Processing (EMNLP), 2009, Print ISBN 978-1-932432-62-6.

[34] A. Levenberg, C. Callison-Burch et M. Osborne, Stream-based translation models for statistical machine translation, International Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), 2010, Print ISBN 1-932432-65-5.

[35] S. Petrovic, M. Osborne et V. Lavrenko, Streaming first story detection with application to Twitter, International Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), 2010, Print ISBN 1-932432-65-5.

[36] M. Isard, M. Budiu, Y. Yu, A. Birrell et D. Fetterly, Dryad: Distributed data-parallel programs from sequential building blocks, ACM European Conference on Computer Systems (EuroSys), 2007, Print ISBN 978-1-59593-636-3.

[37] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda et J. Currey, DryadLINQ: A system for general purpose distributed data-parallel computing using a high-level language, International Symposium on Operating System Design and Implementation (OSDI), 2008, Print ISBN 978-1-60558-551-2.

[38] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser et G. Czajkowski, Pregel: A system for large-scale graph processing, ACM Symposium on Principles of Distributed Computing (PODC), 2009, Print ISBN 978-1-4503-0032-2.

[39] L. G. Valiant, A bridging model for parallel computation, Communications of the ACM, 1990, DOI 10.1145/79173.79181.

[40] H. C. Yang, A. Dasdan, R. L. Hasiao et D. S. Parker, MapReduce-Merge: Simplified relational data processing on large clusters, ACM Conference on Management of Data (SIGMOD), 2007, Print ISBN 978-1-59593-686-8.

[41] J. Hammerbacher, Information platforms and the rise of the data scientist, O'Reilly, 2009, Print ISBN 978-0-59615-711-1.

[42] J. Lin et M. Schatz, Design Patterns for Efficient Graph Algorithms in MapReduce, International Workshop on Mining and Learning with Graphs (MLG), 2010, Print ISBN 978-1-4503-0214-2.

[43] J. Lin, C. Dyer et G. Hirst, Data-Intensive Text Processing with MapReduce (Synthesis Lectures on Human Language Technologies), Morgan and Claypool Publishers, 2010, Print ISBN 978-1608453429.

[44] F. N. Afrati et J. D. Ullman, Optimizing joins in a map-reduce environment, International Conference on Extending Database Technology (EDBT), 2010, Print ISBN 978-1-60558-945-9.

[45] J. Chandar, Join Algorithms using Map/Reduce., M.S. thesis, School of Informatics, University of Edinburgh, 2010.

[46] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Y. Yu, G. Bradski, A. Y. Ng et K. Olukotun, Map-Reduce for Machine Learning on Multicore, International Conference on Advances in Neural Information Processing Systems (NIPS), 2006, Print ISBN 978-0-2621-9568-3.

[47] J. Tordable, MapReduce for Integer Factorization, Computer Science, Cornell University of New York, 2010.

[48] S. Seo, E. J. Yoon, K. Jaehong, J. Seongwook et M. Seungryoul, HAMA: An Efficient Matrix Computation with the MapReduce Framework, International Conference on Cloud Computing Technology and Science (CloudCom), 2010, Print ISBN 978-1-4244-9405-7.

[49] J. Norstad, A MapReduce Algorithm for Matrix Multiplication, Academic and Research Technologies, Northwestern University of Chicago, 2011.

[50] M. Singh, G. Mehta, C. Vaid et P. Oberoi, Detection of Malicious Node in Wireless Sensor Network Based on Data Mining, International Conference on Computing Sciences (ICCS), 2012, Print ISBN 978-1-4673-2647-6.

[51] J. Biskup et L. Li, On Inference-Proof View Processing of XML Documents, IEEE Transactions on Dependable and Secure Computing (TDSC), 2013, ISSN 1545-5971.

[52] S. Janakiraman, S. Rajasoundaran et P. Narayanasamy, The Model - Dynamic and Flexible Intrusion Detection Protocol for high error rate Wireless Sensor Networks based on data flow, International Conference on Computing, Communication and Applications (ICCCA), 2012, Print ISBN 978-1-4673-0270-8.

[53] V. V. Tan, D. S. Yoo et M. J. Myeong-Jae Yi, Design and Implementation of Web Service by Using OPC XML-DA and OPC Complex Data for Automation and Control Systems, IEEE International

Conference on Computer and Information Technology (CIT), 2006, Print ISBN 0-7695-2687-X.

[54] M. K. Ramanathan, A. Grama et S. Jagannathan, Static specification inference using predicate mining, ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2007, Print ISBN 978-1-59593-633-2.

[55] S. V. Stankovic, G. Rakocevic et V. Milutinovic, A metadata-supported distributed approach for data mining based prediction in wireless sensor networks, International Conference on Telecommunication in Modern Satellite Cable and Broadcasting Services (TELSIKS), 2011, Print ISBN 978-1-4577-2018-5.

[56] R. Yanna, L. Suhong et W. Qiang, The design of algorithm for data mining system used for Web Service, IEEE International Conference on Communication Software and Networks (ICCSN), 2011, Print ISBN 978-1-61284-485-5.

[57] V. D. Mabonzo et Z. Weishi, Data mining with inference networks, IEEE International Conference on Communication Software and Networks (ICCSN), 2011, Print ISBN 978-1-61284-485-5.

[58] F. Jiang, C. K. S. Leung et S. K. Tanbeer, Finding Popular Friends in Social Networks, International Conference on Cloud and Green Computing (CGC), 2012, Print ISBN 978-1-4673-3027-5.

[59] N. Salamanos, E. Voudigari, T. Papageorgiou et M. Vazirgiannis, Discovering Correlation between Communities and Likes in Facebook, IEEE International Conference on Green Computing and Communications (GreenCom), 2012, Print ISBN 978-1-4673-5146-1.

[60] A. Hassan, A. Abbasi et D. Zeng, Twitter Sentiment Analysis: A Bootstrap Ensemble Framework, International Conference on Social Computing (SocialCom), 2013, INSPEC Accession 14024707.

[61] S. Shahheidari, H. Dong et M. N. R. Bin Daud, Twitter Sentiment Mining: A Multi Domain Analysis, International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), 2013, Print ISBN 978-0-7695-4992-7.

[62] Y. Tyshchuk, H. Li, H. Ji et W. A. Wallace, Evolution of communities on Twitter and the role of their leaders during emergencies, IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 2013, INSPEC Accession 14222185.

[63] W. Chen, I. Paik, T. Tanaka et B. T. G. S. Kumura, Awareness of Social Influence for Service Recommendation, IEEE International Conference on Services Computing (SCC), 2013, Print ISBN 978-0-7695-5026-8.

[64] L. Guo, X. Que, Y. Cui, W. Wang et S. Cheng, A hybrid social search model based on the user's online social networks, IEEE International Conference on Cloud Computing and Intelligent Systems (CCIS), 2012, Print ISBN 978-1-4673-1855-6.

[65] S. Stieglitz et L. Dang-Xuan, Political Communication and Influence through Microblogging-An Empirical Analysis of Sentiment in Twitter Messages and Retweet Behavior, Hawaii International Conference on System Science (HICSS), 2012, Print ISBN 978-1-4577-1925-7.

[66] S. Triki, N. Bellamine Ben Saoud, J. Dugdale et C. Hanachi, Coupling Case Based Reasoning and Process Mining for a Web Based Crisis Management Decision Support System, IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013, Print ISBN 978-1-4799-0405-1.

[67] A. Tapiador, D. Carrera et J. Salvachua, Social Stream, a social network framework, International Conference on Future Generation Communication Technology (FGCT), 2012, Print ISBN 978-1-4673-5859-0.

[68] Z. Zheng, J. Zhu et M. R. Lyu, Service-Generated Big Data and Big Data-as-a-Service: An Overview, IEEE International Congress on Big Data (BigData Congress), 2013, Print ISBN 978-0-7695-5006-0.

[69] M. Nemiche, V. Cavero et R. Pla Lopez, Understanding social behavior evolutions through agent-based modeling, International Conference on Multimedia Computing and Systems (ICMCS), 2012, Print ISBN 978-1-4673-1518-0.

[70] L. Tan, S. Ponnam, P. Gillham, B. Edwards et E. Johnson, Analyzing the impact of social media on social movements: A computational study on Twitter and the Occupy Wall Street movement, IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 2013, INSPEC Accession 14222194.

[71] T. Mahmood, T. Iqbal, F. Amin, W. Lohanna et A. Mustafa, Mining Twitter big data to predict 2013 Pakistan election winner, International Multi Topic Conference (INMIC), 2013, INSPEC Accession 14079701.

[72] A. Balar, N. Malviya, S. Prasad et A. Gangurde, Forecasting consumer behavior with innovative value proposition for organizations using big data analytics, IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2013, Print ISBN 978-1-4799-1594-1.

[73] L. Verma, S. Srinivasan et V. Sapra, Integration of rule based and case based reasoning system to support decision-making, International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014, INSPEC Accession 14210912.

[74] K. H. Lee, A. Lippman, A. S. Pentland et P. Maes, Just-in-Time Social Cloud: Computational Social Platform to Guide People's Just-in-Time Decisions, IEEE International Conference on Green Computing and Communications (GreenCom), 2013, INSPEC Accession 13972454.

[75] J. Fueller, R. Schroll, S. Dennhardt et K. Hutter, Social Brand Value and the Value Enhancing Role of Social Media Relationships for Brands, Hawaii International Conference on System Science (HICSS), 2012, Print ISBN 978-1-4577-1925-7.

[76] W. Suraworachet, S. Premsiri et N. Cooharojananone, The Study on the Effect of Facebook's Social Network Features toward Intention to Buy on F-commerce in Thailand, IEEE/IPSJ International Symposium on Applications and the Internet (SAINT), 2012, Print ISBN 978-1-4673-2001-6.

[77] M. A. Shaikh et W. Jiaxin, Investigative Data Mining: Identifying Key Nodes in Terrorist Networks, IEEE International Conference on Multitopic Conference (INMIC), 2006, Print ISBN 1-4244-0795-8.

[78] G. Boggia, P. Camarda, L. A. Grieco et M. R. Palattella, Fire Detection using Wireless Sensor Networks: An Approach Based on Statistical Data Modeling, International Conference on New Technologies, Mobility and Security (NTMS), 2008, Print ISBN 978-1-42443547-0.

[79] A. Kulakov et D. Davcev, Distributed data processing in wireless sensor networks based on artificial neural-networks algorithms, IEEE Symposium on Computers and Communications (ISCC), 2005, Print ISBN 0-7695-2373-0.

[80] S. Kim, W. Jung et H. S. Kim, A location inference algorithm based-on smart phone user data modelling, International Conference on Advanced Communication Technology (ICACT), 2014, Print ISBN 978-89-968650-2-5.

[81] S. Gupta et M. Dave, Distributed clustering approach for wireless sensor network based cellular data placement model, IEEE International Conference on Advance Computing Conference (IACC), 2010, Print ISBN 978-1-4244-4790-9.

[82] M. R. Huq, P. M. G. Apers et A. Wombacher, An Inference-Based Framework to Manage Data Provenance in Geoscience Applications, IEEE Transactions on Geoscience and Remote Sensing (TGRS), 2013, ISSN 0196-2892.

[83] H. Yang et S. Fong, Countering the Concept-Drift Problem in Big Data Using iOVFDT, IEEE International Congress on BigData (BigData Congress), 2013, Print ISBN 978-0-7695-5006-0.

Liste des abréviations

A	AaaS	Analytics as a Service (outils d'analyse en tant que service)
	ACID	Atomicity, Consistency, Isolation, Durability (atomicité, consistance, isolation, durabilité)
	AGL	Atelier de Génie Logiciel
	API	Application Programming Interface (interface de programmation)
B	BC	Base de Connaissance
	BI	Business Intelligence (informatique décisionnelle)
	BSON	Binary JSON (JSON binaire)
	BSP	Bulk Synchronous Parallel (synchronisation parallèle en bloc)
C	CAP	Consistency, Availability, Partition Tolerance (CDP)
	CBR	Case-Based Reasoning (raisonnement par étude de cas)
	CDP	Consistance, Disponibilité, Persistance au morcellement
	CPU	Central Processing Unit (unité centrale)
	CRM	Customer Relationship Management (gestion de la relation client)
	CRUD	Create Read Update Delete (créer, afficher, mettre-à-jour, supprimer)
D	DWH	Data Warehouse (entrepôt de données)
E	EM	Expectation-Maximisation (méthode d'estimation dans le cadre du maximum de vraisemblance)
	EO	Exaoctet
	ES	Expert System (SE)
	ETL	Extract-Transform-Load (synchronisation massive d'information)
F	FAI	Fournisseurs d'accès Internet
G	GAFA	Les géants actuels les plus connus du Web (Google, Apple, Facebook, Amazon)
	Gbit/s	Gigabit (débit de transfert réseau)
	GC	Garbage Collector (ramasse-miettes ou récupérateur de mémoire)
	GFS	Google File System (système de fichiers de Google)
	GHZ	Gigahertz (fréquence du processeur de l'unité centrale)
	GNU	GNU's Not UNIX (projet de système d'exploitation Open Source)
	GO	Gigaoctet
	GZIP	GNU ZIP (format de compression de données)
H	HBase	Hadoop Database (SGBD non-relationnel distribué)
	HDD	Hard Disc Drive (disque dur)
	HDFS	Hadoop Distributed File System (système de fichiers distribué de Hadoop)
	HMM	Hidden Markov Model (modèle de Markov caché)
I		
J	JDBC	Java DataBase Connectivity (une interface de programmation permettant d'accéder aux sources de données en utilisant la plate-forme Java)
	JSON	JavaScript Object Notation (notation d'objet JavaScript)
K		

L	LZO	Lempel–Ziv–Oberhumer (format de compression de données)
M	MDA	Model Driven Architecture (architecture dirigée par les modèles)
	MO	Megaoctet
	MPI	Message Passing Interface (interface d'exploration par passage de messages)
	MPP	Massively Parallel Processing (traitement massivement parallèle)
	MVCC	Multiversion Concurrency Control (contrôle des accès concurrents)
N	NoSQL	Not Only SQL (SQL non-relationnel)
	NP	Non-déterministe Polynomial
	NP-complet	Complet pour la classe NP
O	OLAP	Online Analytical Processing (traitement analytique en ligne)
	OpenMP	Open Multi-Processing (interface de programmation pour le calcul parallèle)
P	PDF	Portable Document Format (format portable du document)
	PO	Petaoctet
	PR	PageRank (algorithme d'analyse des liens concourant au système de classement des pages Web)
Q		
R	RAM	Random Access Memory (mémoire vive)
	RDD	Resilient Distributed Dataset (une interface de programmation centrée sur les structures de données)
	RDF	Resource Description Framework (modèle de graphe qui décrit les ressources Web)
	REST	Representational State Transfer (modèle décrivant le fonctionnement du Web)
	RR	Round-Robin Algorithm (algorithme d'ordonnance)
S	SaaS	Software as a Service (logiciel en tant que service)
	SE	Système Expert
	SGBD	Systèmes de Gestion des Bases de Données
	SLA	Service Level Agreement (qualité de service)
	SQL	Structured Query Language (langage de requêtes structurées)
T	TAR	Tape Archiver (format de compression de données)
	TO	Teraoctet
	TTL	Time To Live (durée de vie)
U		
V		
W		
X	XML	Extensible Markup Language (langage de balisage extensible)
Y	YAML	YAML Ain't Markup Language (représentation des données par sérialisation Unicode)
	YARN	Yet Another Resource Negociator (nouvelle version de MapReduce)
Z	ZIP	Format de compression de données

Annexes

Annexe 1. La solution BigQuery de Google

1.1 Introduction

Google BigQuery est une solution de type AaaS qui repose sur la plate-forme Cloud de Google et de ce fait sur sa puissance de calcul. Grâce à BigQuery on peut stocker, effectuer des requêtes et analyser des grands volumes de données. Requêter des tables de plusieurs TO / PO de données, ne prend que quelques secondes. Cette solution s'intègre bien avec Google App Engine et avec d'autres plate-formes. Techniquement, le service BigQuery est juste un serveur qui accepte les requêtes HTTP et renvoie les réponses au format JSON, comme indiqué dans la Figure 62.

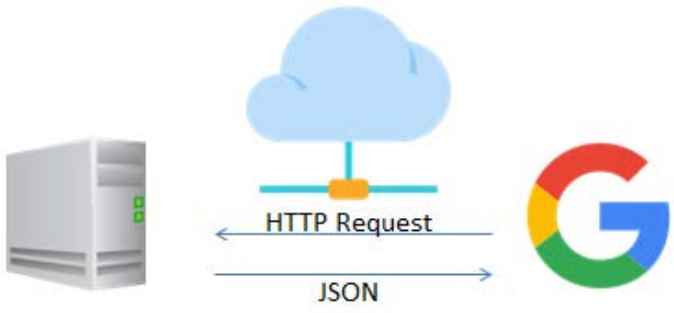

Figure 62 : Modèle de communication BigQuery

1.1.1 L'historique de BigQuery

Comme beaucoup d'outils et de services, BigQuery a été conçu pour résoudre un problème. Les ingénieurs de Google avaient du mal à suivre le rythme de croissance de leurs données. Le nombre d'utilisateurs de Gmail augmentait constamment et était de l'ordre de centaines de millions. En 2012, il y avait plus de 100 milliards de demandes de recherche Google effectuées chaque mois. Essayer de donner un sens à toutes ces données, prenait un temps fou et était une expérience très frustrante pour les équipes de Google.

Ce problème de données a conduit à l'élaboration d'un outil interne appelé Dremel, qui a permis aux employés de Google d'exécuter des requêtes SQL extrêmement rapides, sur un grand ensemble de données. Dremel est devenu extrêmement populaire chez Google. Les ingénieurs de Google l'utilisent des millions de fois par jour. Le moteur de requêtes Dremel, a créé une façon de paralléliser l'exécution des requêtes SQL, sur des milliers de machines.

Dremel peut scanner 35 milliards de lignes sans un index et en une dizaine de secondes. Il existe deux technologies que Dremel utilise pour atteindre la lecture d'1 TO de données en quelques secondes :

1- Le premier est appelé Colossus. Il s'agit d'un système de fichiers distribués et parallélisables, développé chez Google, comme un successeur de GFS.
2- Le second est le format de stockage, appelé ColumnIO, qui organise les données d'une manière à ce que ce soit plus facile de les interroger.

En 2012, Google a lancé publiquement BigQuery qui a permis aux utilisateurs en dehors de Google de profiter de la puissance et la performance de Dremel. BigQuery communique avec Dremel, le moteur de requêtes qui communique avec Colossus.

1.2 Les critères d'analyse de BigQuery

Plusieurs critères sont en vigeur afin d'étudier cette solution :

1- Les avantages.
2- Les inconvénients.
3- Les quotas.
4- Le mode de facturation.

1.2.1 Les avantages de BigQuery

BigQuery présente une longue liste d'avantages :

1- Le mode de communication sécurisé.
2- Le niveau SLA à 99.9%.
3- Le principe d'utiliser l'infrastructure d'un géant, tel que Google.
4- L'absence de coût de serveurs, d'opération et de maintenance.
5- La simplicité comparée à l'écosystème Hadoop.
6- L'utilisation des requêtes SQL.
7- La stabilité et les temps de réponse.
8- La facturation au temps d'utilisation.
9- La possibilité de traiter des requêtes synchrones ou asynchrones.
10- La facilité d'interconnexion avec des outils tiers.

1.2.2 Les inconvénients de BigQuery

BigQuery présente également certains inconvénients :

1- On ne peut pas modifier ou supprimer des entrées dans une table, seulement y rajouter des données.
2- On n'est pas à l'abri des latences réseau.
3- BigQuery est moins performant sur les petites tables.

1.2.3 Les quotas

Seules 20000 requêtes sont autorisées par jour ou l'équivalent de 100 TO de données. Au moment du chargement ou téléchargement des données :

1- Une limite quotidienne de 1000 Jobs de chargement par table s'impose, ainsi que 10000 Jobs de chargement par projet, y compris les échecs.
2- Seuls 1000 fichiers par Job de chargement sont autorisés.
3- La taille maximale d'un fichier est limitée selon le type, comme indiqué dans le Tableau 14.

Type de fichier	Compressé	Non compressé	Condition
CSV	4 GO	4 GO	Avec de nouvelles lignes dans les chaînes de caractères
		5 TO	Sans nouvelles lignes dans les chaînes de caractères
JSON	4 GO	5 TO	

Tableau 14 : Limitations de BigQuery en taille de fichiers

En mode flux de données, seules 100000 lignes insérées par seconde par table sont autorisées. A noter que toutes les données qui sont en cache, ne sont pas comptées dans les quotas.

1.2.4 Le mode de facturation

Les utilisateurs de BigQuery sont actuellement facturés pour 2 choses, le stockage et les requêtes. Les deux coûts sont proportionnels à la taille des données. Importer des données, via un fichier CSV ou JSON reste cependant gratuit.

1.3 L'architecture technique

BigQuery est une implémentation externe de Dremel. Pour comprendre comment cela fonctionne, il faut connaître les deux technologies de base de Dremel.

1.3.1 L'architecture en arbre

Ce type d'architecture est utilisé pour dispatcher les requêtes et agréger les résultats à travers des milliers de machines en quelques secondes, comme illustré dans la Figure 63.

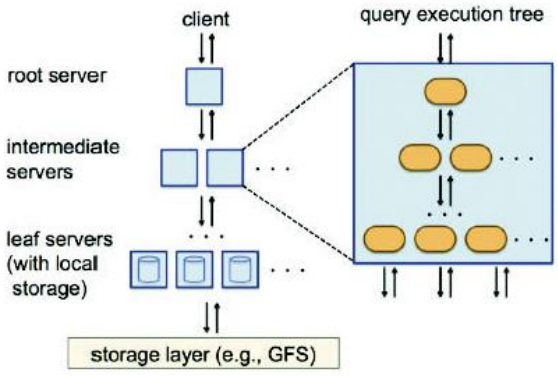

Figure 63 : Architecture en arbre de Dremel

1.3.2 La base de données orientée colonne

Dremel utilise BigTable et ColumnIO, des bases de données orientées colonne. Il stocke ensuite les données sous forme de colonnes. Grâce à cela, on ne charge que les colonnes dont on a vraiment besoin. Dremel stocke les enregistrements par colonne sur des volumes de stockage différents, alors que les bases de données traditionnelles, stockent normalement l'ensemble des données sur un seul volume. Cela permet un taux de compression très élevé. Le mécanisme de stockage Dremel es illustré dans la Figure 64.

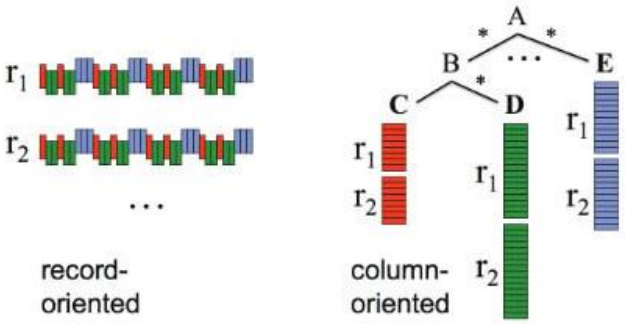

Figure 64 : Modèle orienté colonne de Dremel

1.4 Les composants de BigQuery

Il existe 3 composants principaux dans BigQuery.

1.4.1 Les projets

Toutes les données dans BigQuery sont stockées dans des projets. Un projet est identifié par un identifiant unique, ainsi qu'un nom. Il contient la liste des utilisateurs autorisés, les informations concernant la facturation et l'authentification à l'API.

1.4.2 Les Datasets

Les tables sont groupées par Dataset. Un Dataset peut être partagé à d'autres utilisateurs. Il peut contenir une ou plusieurs tables.

1.4.3 Les tables

Les tables sont représentées de la façon suivante :

$< project >:< dataset >.< table\_name >$

Pour comprendre cette règle de nommage, Il faut regarder de plus près un exemple de requête SQL :

$SELECT\ date, title\ FROM\ [hadoop: 12930. code]$

Les champs « date » et « title » sont sélectionnés dans la table « code » qui est dans le Dataset « 12930 » appartenant au projet « hadoop ».

Lorsqu'on crée une table il faut définir son schéma :

$$date: TIMESTAMP, author: STRING, title: STRING, word\_count: INTEGER$$

Le résultat est illustré dans la Figure 65.

Schema

date	TIMESTAMP	REQUIRED	Describe this field...
author	STRING	REQUIRED	Describe this field...
title	STRING	REQUIRED	Describe this field...
word_count	INTEGER	NULLABLE	Describe this field...

Figure 65 : Exemple de schéma de table BigQuery

1.5 Le mode d'accès à BigQuery

Plusieurs modes de communication sont possibles afin d'accéder à BigQuery :

1- Via la console Web de l'API de Google. Cette interface permet d'effectuer la plupart des opérations, en commençant par lister les tables, afficher leurs schémas et leurs données ou encore partager des Datasets avec d'autres utilisateurs et pour finir, charger les données dans les tables. Cette console est pratique dans l'exécution, le teste et l'optimisation des requêtes.
2- En ligne de commande (par exemple, pour ajouter une colonne dans une table). Pour ce faire, Python est un prérequis sur la machine.
3- Via les API et les librairies mises à disposition par Google, telles que les librairies disponibles pour Java, Python, PHP, Ruby, .Net, REST ou autres. Toutes ces API utilisent le mode d'authentification OAuth2.
4- Via des connecteurs JDBC, Excel, Hadoop.
5- Via le connecteur pour le nouveau service Google Cloud DataFlow.

1.6 Le chargement des données

Il existe 2 moyens pour charger des données dans une table :

1- Via la console Google, lors de la création de la table, on définit une source de données. Cela peut être un fichier au format CSV, JSON ou AppEngine Datastore, qui peut être chargé ou importé du module Google Cloud Storage. Ensuite, on définit le schéma de cette table.
2- Via l'API disponible. On peut charger un fichier CSV ou JSON mais encore insérer des entrées dans une table.

$List < TableDataInsertAllRequest.Rows > rowList = new$
$ArrayList < TableDataInsertAllRequest.Rows > ();$
$rowList.add(new\ TableDataInsertAllRequest.Rows()$
$.setInsertId("" + System.currentTimeMillis())$
$.setJson(new\ TableRow().set("adt", null)));$
$TableDataInsertAllRequest\ content = new\ TableDataInsertAllRequest().setRows(rowList);$
$TableDataInsertAllResponse\ response = bigquery.tabledata().insertAll(PROJECT\_NUMBER,$
$DATASET\_ID,$
$TABLE\_ID,$
$content).execute();$
$System.out.println("kind = " + response.getKind());$
$System.out.println("errors = " + response.getInsertErrors());$
$System.out.println(response.toPrettyString());$

1.7 BigQuery SQL

Les requêtes sont écrites en utilisant une variante de l'instruction « SQL SELECT » standard. BigQuery prend en charge une grande variété de fonctions, telles que « COUNT », les expressions arithmétiques et les fonctions de chaîne. Tout comme le SQL standard, l'instruction « SELECT » s'écrit de cette manière :

$SELECT\ expr1\ [[AS]\ alias1]\ [, expr2\ [[AS]\ alias2], \dots]$
$[agg\_function(expr3)\ WITHIN\ expr4]$
$[FROM\ [(FLATTEN(table\_name1|(subselect1)]\ [, table\_name2|(subselect2), \dots)]$
$[[INNER|LEFT\ OUTER|CROSS]\ JOIN\ [EACH]\ table\_2|(subselect2)\ [[AS]\ tablealias2]$
$ON\ < em > join\_condition\_1 < /em >\ [\dots\ AND\ < em > join\_condition\_N < /em >\ \dots]] +$
$[WHERE\ condition]$
$[GROUP\ [EACH]\ BY\ field1|alias1\ [, field2|alias2, \dots]]$
$[HAVING\ condition]$
$[ORDER\ BY\ field1|alias1\ [DESC|ASC]\ [, field2|alias2\ [DESC|ASC], \dots]]$
$[LIMIT\ n];$

Par ailleurs, on note quelques fonctionnalités intéressantes de BigQuery SQL, telles que « TABLE_DATE_RANGE et REGEXP_MATCH ».

1.8 Les cas d'emploi

Il existe plusieurs solutions pour visualiser les données qui sont stockées dans BigQuery :

1- Utiliser Google App Scripts pour écrire les requêtes et visualiser les résultats dans des graphiques.
2- Développer une application Web / mobile, via les API et librairies mises à disposition afin de visualiser les données avec l'API Google Charts par exemple.

La Figure 66 et la Figure 67 illustrent des exemples de cas d'emploi de BigQuery.

Figure 66 : Exemple des indicateurs d'utilisation de BigQuery

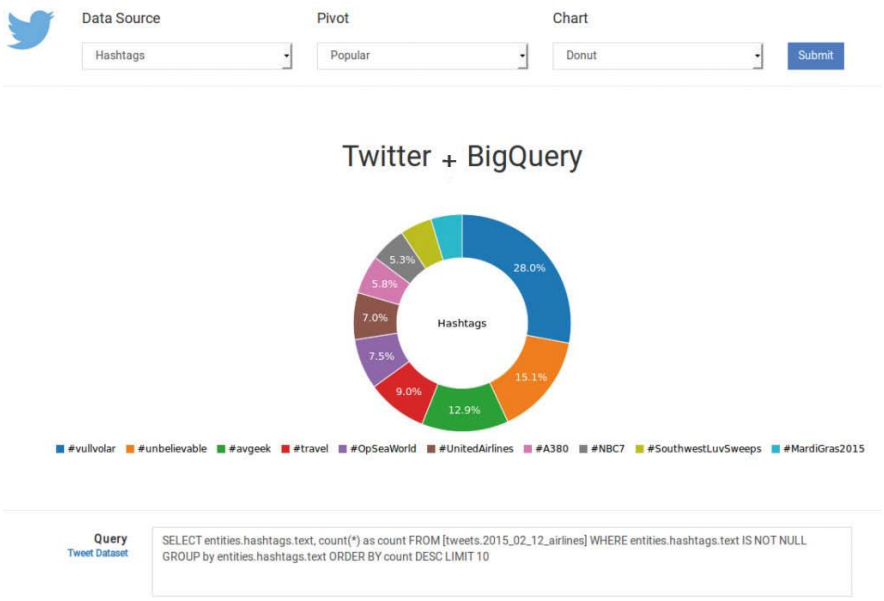

Figure 67 : Traitement des données Twitter avec BigQuery

1.9 Conclusion

BigQuery n'est pas une solution miracle mais elle répond à beaucoup de besoins des utilisateurs. Son côté pratique, en termes d'interrogation de la base de données avec des instructions SQL, est mis en avant. En quelques secondes, il est possible d'exécuter sa requête et obtenir son rapport exporté en CSV, tout en profitant de l'infrastructure de Google.

Annexe 2. L'évaluation du modèle de données orienté document de NoSQL

2.1 Introduction

L'objectif de cette annexe est de présenter une évaluation du modèle orienté document à l'aide d'un ensemble de critères prédéfinis. On évoquera en parallèle MongoDB, la base de données NoSQL qui implémente le modèle orienté document et qui bouscule depuis une dizaine d'années les systèmes de gestion des bases de données classiques. Cette annexe n'a pas pour but de donner des explications sur l'installation et la configuration du logiciel MongoDB. Des commandes utilisées avec ce logiciel, sont toutefois proposées à titre d'exemples dans les paragraphes d'évaluation des critères.

2.2 Le modèle orienté document

Le modèle orienté document est représenté dans la Figure 68. Dans ce modèle, une clé est associée à un document, ce document est constitué d'un ensemble de champs imbriqués. Chacun de ces champs peut être indexé individuellement. Les documents sont regroupés au sein d'une collection.

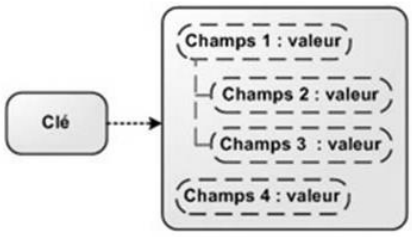

Figure 68 : Modèle orienté document

Une base de données orientée document contient un ensemble de collections, ces collections seront matérialisées lorsque des données seront ajoutées dans la base. Les logiciels les plus répandus implémentant le modèle orienté document, sont CouchDB et MongoDB. La structure imbriquée du modèle orienté document, se rapproche de celle du XML. Cette structure a l'avantage d'être compatible avec les systèmes à objets.

MongoDB est un logiciel très populaire. De nombreuses compagnies, telles que ebay, Forbes, Bosch, Doodle, le CERN l'ont implémenté. Les données sont stockées sous la forme d'objets binaires dans des fichiers de type BSON. La communication entre l'utilisateur et la base, se fait à l'aide d'un Shell Javascript.

2.3 Les critères d'évaluation du modèle

Afin d'évaluer le modèle orienté document, un certain nombre de critères est indispensable.

2.3.1 La nature des données

Les données se situent en général entre des données structurées et non-structurées ou faiblement structurées.

2.3.1.1 Les données structurées

Les bases de données orientées document, telles que MongoDB, stockent des documents au format JSON. JSON est un format de définition de données qui contient une structure hiérarchique de propriétés et de valeurs. C'est un standard ouvert, humainement lisible. Avec le XML, JSON est le format principal pour l'échange de données, utilisé sur le Web moderne. Comparativement à XML, JSON est plus compact. JSON prend en charge tous les types de données de base (les nombres, les chaînes et les valeurs booléennes, ainsi que des tableaux). L'exemple suivant représente un document au format JSON :

```
{"firstName": "Hadi",
"lastName": "Hashem",
"age": 32,
"address": {"streetAddress": "9 Rue Charles Fourier", "city": "Evry", "state": "Essonne",
            "postalCode": "91000"},
"phoneNumber": [{"type": "personal", "number": "0623456789"},
                {"type": "fax", "number": "0923456789"}]}
```

2.3.1.2 Les données faiblement structurées

Le format JSON remplit le même cahier des charges que le XML, en tout cas sur le principe. Il est donc taillé pour stocker des données semi-structurées. Des documents, contenant du texte libre, peuvent être également stockés, sous forme de couples clé-valeur. Un index configuré sur ce champ permet de rechercher une chaîne de caractères à l'intérieur du texte.

2.3.2 La relation

Dans un modèle relationnel, une base de données consiste en une ou plusieurs relations. Les lignes de ces relations sont appelées des tuples ou enregistrements. Les noms des colonnes sont appelés des attributs.

2.3.2.1 L'intégrité référentielle

Le modèle orienté document est fortement dé-normalisé, un document contient toute les informations liées à une entité. Pour pouvoir réaliser des relations dans ce modèle, il faut ajouter l'identifiant d'un document dans un autre document, ce qui revient à créer une clé étrangère. C'est l'application qui doit gérer ces identifiants, pour pouvoir mettre les documents en relation. Dans l'exemple proposé dans la Figure 69, on peut connaître les produits qui ont été achetés par « Alice ». En revanche, la réciproque est plus complexe, si l'on souhaite connaître qui a acheté le produit « crème glacée », il est nécessaire de parcourir tous les utilisateurs ou créer des éléments de type Back Références.

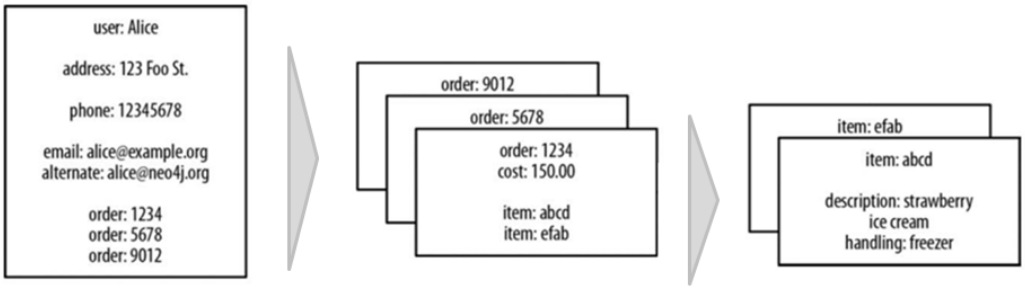

Figure 69 : Références entre documents

L'application doit gérer également les références lorsque des documents sont supprimés.

2.3.2.2 Les relations hiérarchiques

De par sa nature, un document JSON est bien adapté pour structurer des données hiérarchiques.

2.3.3 Le cycle de vie

Le cycle de vie des données relève de la gestion des versions et de la durée de vie.

2.3.3.1 La gestion des versions

Dans le modèle relationnel, si la structure d'une table doit être modifiée dans le but de faire évoluer une application et qu'il est nécessaire de conserver l'ancienne structure, la table d'origine devra être renommée, afin que le nom de cette table contienne un numéro de version. Dans le modèle orienté document, grâce à la souplesse de sa structure, c'est beaucoup plus simple, il suffit de rajouter un champ dans le document pour le numéro de version.

2.3.3.2 La durée de vie

Il est possible d'associer une durée de vie, dite TTL, à un document. Le GC va se charger de supprimer les documents à l'échéance du délai. Ce GC est exécuté toutes les 60 secondes. Cela signifie que certains documents peuvent exister encore quelques secondes après avoir expiré. Pour configurer un TTL, il faut créer un index sur un champ de type date. La commande suivant permet de supprimer les données de session d'un utilisateur après une heure :

$$db.sessions.ensureIndex(\{\ "timestamp": 1\ \},\{\ expireAfterSeconds: 3600\ \})$$

Lorsque le temps est expiré, c'est tout le document qui est supprimé. Il est possible également, de configurer une date à laquelle le document sera éliminé.

2.3.4 Le schéma et les opérations CRUD

Dans un SGBD, les tables sont définies par un schéma structurant les données stockées, sur lesquelles agissent les opérations CRUD.

2.3.4.1 Le schéma

Le modèle orienté document permet de manipuler des objets sans schéma prédéterminé. Dans Mongo, il n'y a pas de tables ni de colonnes et on ne spécifie pas de type pour les données. Les documents sont stockés dans des collections.

Par analogie avec le modèle relationnel, les collections sont l'équivalent des tables et les documents sont l'équivalent des tuples. Les documents stockés dans une collection, peuvent avoir un nombre de champs qui varie. En règle générale, les documents seront stockés avec la même structure mais la souplesse du schéma est un des avantages de ce modèle.

2.3.4.2 Les opérations CRUD

La manipulation des données contenues dans une collection se fait à l'aide des fonctions. La syntaxe est utilisée comme suivant:

$db.NomDeCollection.NomDeFonction()$

Pour insérer une donnée, on utilise la fonction insert.

$db.sessions.insert(\{id: 1234, nom: 'HASHEM', prenom: 'HADI', timestamp : new\ Date()\})$

Pour chaque document inséré dans une collection, MongoDB génère un identifiant unique. Cet identifiant joue le rôle de clé primaire, il permet d'identifier de façon unique un document dans une collection.

Pour récupérer tous les documents d'une collection on utilise la fonction « find ».

$db.sessions.find()$

Les mises-à-jours sur un document peuvent s'effectuer de 2 manières :

1- Un document peut être remplacé par un nouveau document.
2- L'opérateur « $set » permet de faire une mise-à-jour partielle du document sans écraser le document d'origine.

$db.sessions.update(\{nom: 'HASHEM'\}, \{\ languages: ['Java', 'c', 'c++']\})$
$db.sessions.update(\{nom: 'HASHEM'\}, \{\$set: \{'taille': 185\}\ \})$

Pour supprimer un document on utilise la fonction « remove ».

$db.sessions.remove(\{nom: 'HASHEM'\})$

En comparaison des autres SGBD NoSQL, MongoDB propose de très nombreuses fonctionnalités pour manipuler des données contenues dans les documents. Cependant, on est encore très loin des possibilités qu'offre le langage SQL.

2.3.5 La consistance des données

La consistance est le deuxième composant des quatre propriétés ACID.

2.3.5.1 Les propriétés ACID

MongoDB est une base conforme aux propriétés ACID, c'est-à-dire que les verrous traditionnels et les transactions impliquant plusieurs documents ne sont pas supportés. En revanche, elle permet d'effectuer des opérations atomiques sur un document. Par exemple, retirer un élément de la collection « stock » pour l'ajouter à la collection « ventes » dans une seule et même transaction n'est pas possible, à moins que « stock » et « vente » ne se trouvent dans le même document.

Pour pallier ce problème on peut imaginer d'utiliser MongoDB en conjonction avec une base qui supporte les transactions, comme Postgress par exemple. Il s'agit dans ce cas d'une approche multi-modèle.

2.3.6 La performance

MongoDB n'a pas de cache configurable, la base utilise automatiquement toute la mémoire à disposition sur la machine par l'intermédiaire de fichiers « map » en mémoire. A la différence des SGBD relationnels qui transforment les données en représentations objets, utilisables par les applications. La représentation d'un document en mémoire dans Mongo, est la même que celle qui se trouve sur le disque. Cette mise en cache est moins coûteuse en termes de ressources. La base garde en mémoire toutes les données récemment utilisées qui peuvent tenir dans la RAM.

2.3.6.1 L'indexation

Par défaut, toutes les collections ont un index sur le champ d'identification du document (cet identifiant technique est codé sur 12 bytes). Il est possible d'ajouter des index secondaires sur n'importe quel champ apparaissant dans le document. Dans la Figure 70, le champ « category » a été indexé pour permettre par exemple d'accélérer une recherche pour retrouver les documents qui ont comme valeur « networking ». La Figure 71 présente la structure d'un index dans MongoDB. Un index est constitué d'une copie du champ à indexer (le champ nom) associé à l'identifiant du document (la clé). Ces index sont appelés index simples. Il existe d'autres types d'index dans MongoDB, comme par exemple les index composés ou les index multi-clés. Un index composé possède des références sur plusieurs champs appartenant à un document. Les index multi-clés permettent d'indexer les éléments d'un tableau.

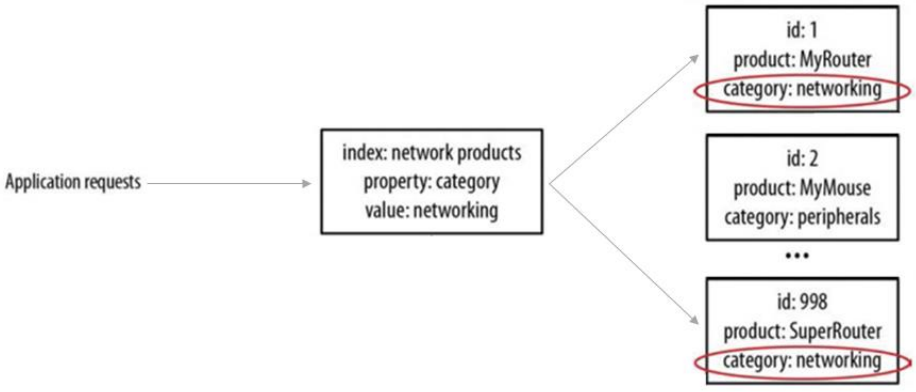

Figure 70 : Indexation dans le modèle orienté document

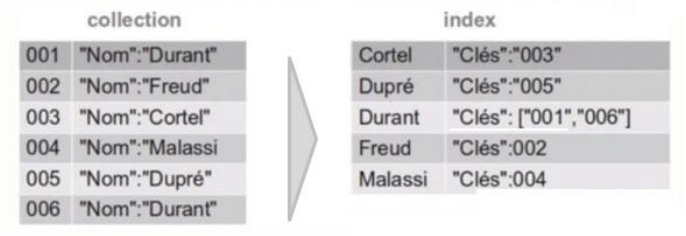

Figure 71 : Structure d'un index dans le modèle orienté document

2.3.6.2 Le partitionnement

MongoDB est construit pour l'évolutivité applicative et la haute disponibilité. Un cluster mongo est composé d'un ensemble de nœuds, possédant chacun leur propre jeu de données. Les données insérées sont automatiquement distribuées sur l'ensemble de ces nœuds, de façon à équilibrer la charge. La Figure 72 montre un exemple d'architecture distribuée MongoDB. Le Config Server contient toutes les informations liées à la topographie du Cluster. Mongos est le processus coordinateur. Le Deamon attend les connexions des applications et distribue les écritures sur les nœuds du Cluster. Les partitions, dits Shard, sont les serveurs membres. Étant donné que les données sont répliquées sur plusieurs nœuds, les requêtes sont distribuées sur un ensemble de serveurs, par conséquent, garantissent un temps de réponse optimal. La distribution des données permet de répondre à une augmentation significative du volume des données à traiter.

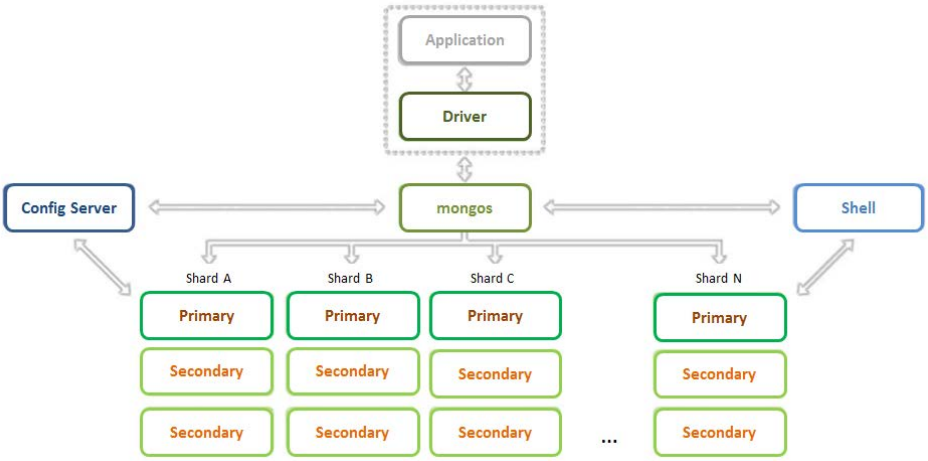

Figure 72 : Cluster MongoDB

2.3.7 La volumétrie

L'étendue de l'utilisation des téléphones portables, la multiplication des différents appareillages de recueil d'information (stations météo, télescope, séquenceurs d'ADN, données de flux RSS) produisent des volumes de données gigantesques. De nombreuses entreprises utilisent MongoDB pour ses capacités à stocker de grands volumes d'information.

2.3.8 L'agrégation des données

MongoDB implémente l'algorithme MapReduce. Cet algorithme effectue des opérations d'agrégations sur des collections. On prend un exemple pour expliquer son fonctionnement. Le Tableau 15 représente la collection « COMMANDES ». Cette collection contient l'ensemble des produits vendus par une entreprise. Chaque ligne correspond à un document Mongo. On veut calculer le montant total pour chaque produit.

no_comm	id_client	id_article	quantite	prix
25	AA	X	3	20
34	CC	Y	2	40
44	CC	Z	1	10
36	AA	Y	5	40
40	BB	Z	7	11
44	CC	X	2	20
48	AA	Y	1	40

Tableau 15 : Exemple de collection COMMANDE dans le modèle orienté document

La fonction « map » prend en paramètre un couple clé-valeur, la clé correspond à l'identifiant des articles, la valeur est une opération « prix x quantite ».

```
var mapf = function() {
    emit(this.id_article, this.quantite*this.prix);
}
```

Le Tableau 16 montre le résultat obtenu avec la fonction « map ». Les données sont triées par produit.

Id_article	total
X	60
X	40
Y	80
Y	200
Y	40
Z	87

Tableau 16 : Résultat de la fonction map sur les données de la collection COMMANDE

La fonction « reduce » prend en paramètre l'identifiant des articles et le champ à agréger.

```
var reducef = function(id_article, montant){
    return Array.sum(montant);
}
```

Le résultat obtenu est affiché dans le Tableau 17.

Id_article	total
X	100
Y	320
Z	87

Tableau 17 : Résultat de la fonction reduce sur les données de la collection COMMANDE

La commande suivant va déclencher les calculs.

$$db.ordres.mapReduce(mapf, reducef, \{out: "map\_reduce\_commandes"\})$$

La sortie est renvoyée dans une collection. Pour obtenir le résultat, il suffit d'afficher le contenu de la collection « map_reduce_commandes ».

2.3.9 La persistance et la résilience

Si une base Mongo n'est pas arrêtée correctement, suite à un défaut d'alimentation par exemple, alors cette base se retrouve dans un état inconsistant, c'est-à-dire que les dernières écritures sur la base n'ont pas encore été enregistrées dans les documents. Dans mongo, les écritures sont poussées dans un fichier journal toutes les 100 millisecondes. Cela signifie que les écritures sont persistantes et qu'une instance Mongo est capable d'effectuer une reprise sur panne, dite Instance Crash Recovery.

2.3.9.1 La réplication

La réplication est un processus qui consiste à maintenir plusieurs copies d'une même base de données. La réplication maître-esclave est le mode de réplication supporté par MongoDB. Ce mode est flexible et permet d'augmenter la disponibilité du système. Il peut en outre être utilisé pour faire de la reprise sur panne automatique. Comme il existe plusieurs copies des données, les réplicas protègent le système lors de la perte d'un serveur.

Le principe de fonctionnement de la réplication dans MongDB est défini comme suivant:

1- Une instance primaire (processus mongod) reçoit toutes les opérations.
2- En écriture, les données modifiées sont transférées dans un fichier journal (fichier oplog), les instances secondaires vont mettre à jour leur jeu de données à partir de ce fichier journal.
3- Dans le cas où l'instance primaire tombe, un des membres va prendre le relai à la suite d'une élection. Cette étape est appelée Failover.
4- Il est possible de configurer un arbitre afin de faciliter ce vote. L'élection d'un nouveau membre pour prendre la fonction d'instance primaire est réalisée de façon automatique.

2.3.10 La confidentialité

La confidentialité relève de la gestion des droits et du chiffrement.

2.3.10.1 Les droits d'accès

Par défaut, l'authentification avec mot de passe n'est pas activée sur MongoDB, il est possible de l'activer au démarrage du serveur. Il existe par défaut une base administrateur dans MongoDB. Un utilisateur administrateur possède des privilèges supplémentaires par rapport aux autres utilisateurs, comme celui par exemple d'arrêter le serveur de bases de données. Les droits d'accès aux ressources de la base sont gérés, grâce à des rôles. Les utilisateurs peuvent avoir plusieurs rôles.

2.3.10.2 Le chiffrement

Il n'existe actuellement aucune option pour chiffrer des documents par défaut dans MongoDB. Cependant, il est possible d'inclure des modules de chiffrement, en utilisant des algorithmes Open Source disponibles sur Internet comme Reedb.

2.4 Conclusion

Le Tableau 18 est une synthèse de l'ensemble des critères évoqués pour les deux modèles étudiés précédemment. La couleur rouge signifie un mauvais état, une dégradation des performances ou un critère non implémenté. La couleur jaune signifie une réalisation possible et la couleur verte met en avant les bonnes performances d'un critère ou un critère clé du modèle.

Rubriques	**Catégories**	**Modèle orienté graphe**
Données	Nature des données	Données structurées
		Données faiblement structurées
	Relation entre les données	Intégrité référentielle
		Relations hiérarchiques
	Cycle de vie	Gestion des versions
		Durée de vie
Requêtes	Schéma et opérations CRUD	Schéma
		Opérations CRUD
	Consistance des données	Propriétés ACID
Performance	Performance	Index
		Partitionnement
	Volumétrie	Volumétrie
Analyse	Analyse	Agrégation des données
Persistance	Résilience	Réplication
Sécurité	Confidentialité	Droits d'accès
		Chiffrement

Tableau 18 : Critères de performance du modèle orienté document

Les transactions dans ce modèle ne sont pas prises en charge par le moteur de bases de données. En effet, c'est l'application qui prend à son compte la gestion de l'intégrité des données. Par conséquent, ce modèle a l'avantage d'être beaucoup plus souple que le modèle tabulaire que l'on retrouve dans le monde relationnel. Les Clusters MongoDB offrent de grandes possibilités, en termes de distributions des données et de haute disponibilité. C'est peut-être pour cela, qu'actuellement, MongDB est une des bases de données NoSQL qui rencontre le plus de succès.

Annexe 3. Le modèle orienté graphe de NoSQL comparé au model relationnel

3.1 Introduction

Cette annexe a pour but de décrire les forces et les faiblesses du modèle NoSQL orienté graphe et de le comparer avec le modèle relationnel classique en s'appuyant sur différents critères. On évoquera en parallèle Neo4J, la base de données la plus répandue, qui implémente le modèle orienté graphe.

3.1.1 Les bases de données NoSQL

Le terme NoSQL a été utilisé pour la première fois en 1998. Comme son nom l'indique, il ne s'agit pas de la négation du SQL mais d'une alternative aux bases de données relationnelles. Comme pour d'autres technologies de bases de données possédant leur propre paradigme (base de données XML, base de données objets), la technologie NoSQL possède certaines caractéristiques qui lui sont propres. Dans les années 90, de nouveaux besoins, en termes de stockage des données, sont apparus, une volumétrie des données en expansion liée à l'Internet, des temps de réponse toujours plus critiques et la disponibilité des services qui ne doit souffrir d'aucune interruption. Par conséquent les grands acteurs du Web, comme Google ou Amazon ont fini par développer leur propre solution de base de données, pour stocker des informations récoltées sur toute la toile. Ces nouveaux développements, ont donné naissance à la technologie NoSQL. Les technologies NoSQL implémentent plusieurs modèles de stockage.

Le modèle relationnel a été proposé en 1970 par IBM. C'est un modèle formel capable de décrire, en termes informatiques, les données opérationnelles liées à un système d'information. L'information y est structurée dans un modèle prédéfini, le modèle entité-association dans lequel les données sont représentées sous forme tabulaire. Le modèle relationnel fournit des mécanismes, qui permettent de garantir la consistance des données au sein du système, de gérer les accès concurrents et de garantir les propriétés ACID des transactions. Les données sont manipulées par un langage spécifique pour les bases de données, il s'agit du SQL. C'est un langage déclaratif qui permet de réaliser des opérations sur un ensemble de tables. Il s'agit d'un paradigme de programmation, son objectif est de décrire ce que le programme doit accomplir, plutôt que d'exprimer comment il doit le faire. Le fondement de ce langage repose sur des concepts mathématiques, que l'on retrouve dans l'algèbre relationnel (une théorie mathématique proche de la théorie des ensembles, qui définit des opérations qui peuvent être effectuées sur des relations). Au début des années 80, c'est Oracle qui va proposer la première implémentation commerciale du modèle relationnel. Par la suite, ce modèle va devenir très populaire et beaucoup d'autres constructeurs vont l'implémenter.

Les bases NoSQL sont conçues pour fonctionner dans une architecture distribuées et le paradigme lié à cette technologie vise à optimiser les performances du système à plusieurs niveaux. Les données sont distribuées de manière automatique sur plusieurs serveurs. Afin de garantir les meilleurs temps de réponse possibles, les mécanismes transactionnels sont abandonnés et les données sont dé-normalisées. Pour augmenter la puissance du système, il est possible de rajouter des machines dans le Cluster, ces modifications n'ont pas d'incidence sur la disponibilité des données, on parle de redimensionnement horizontal à chaud. Le langage SQL n'est plus le langage utilisé pour interroger ces bases mais d'autres syntaxes sont utilisées.

3.2 Le modèle orienté graphe

Ce modèle est inspiré de la théorie des graphes. Les données sont modélisées sous la forme de nœud. Les nœuds sont reliés entre eux par des arcs orientés. On retrouve l'implémentation de ce modèle dans les réseaux sociaux pour lesquels on peut intuitivement percevoir l'intérêt d'utiliser un modèle de graphe, un nœud correspond à un utilisateur et les arcs représentent les liens entre ceux-ci. Des algorithmes de parcours de graphe, vont permettre de naviguer, à travers cette structure. Le modèle de graphe possède les propriétés suivantes :

1- Il contient des nœuds et des relations.
2- Les nœuds contiennent des propriétés (couple clé-valeur).
3- Les relations sont nommées et possèdent une direction.
4- Les relations peuvent aussi contenir des propriétés.

Ce modèle présente l'avantage de pouvoir facilement repérer une valeur dans un réseau d'information interconnecté, par exemple pour trouver le produit qui pourrait potentiellement intéresser un client. Dans le graphe de la Figure 73, on a modélisé une application pour une plate-forme d'e-commerce. Ce graphe présente l'historique des commandes pour l'utilisateur « Alice ». Chaque commande contient un ou plusieurs produits. Ce graphe permet également, de connaître la chronologie des achats pour « Alice », grâce aux relations « most_recent » et « previous ».

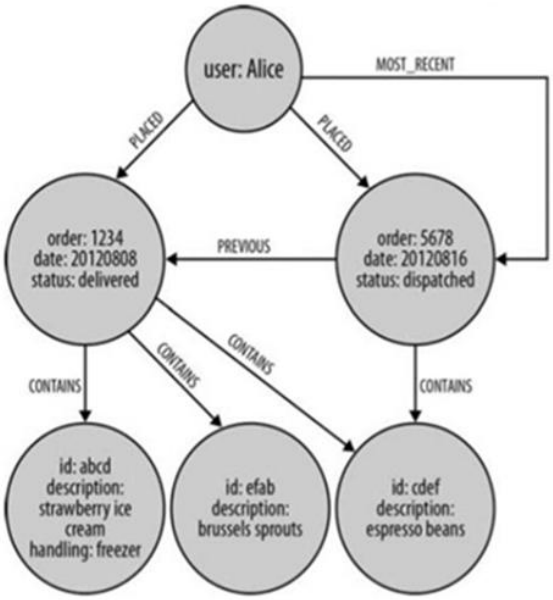

Figure 73 : Historique des commandes de l'utilisateur e-commerce

3.3 Les critères d'évaluation des modèles

Afin d'évaluer le modèle orienté graphe et le modèle relationnel, un certain nombre de critères est indispensable.

3.3.1 La nature des données

Les données se situent en général entre des données structurées et non-structurées ou faiblement structurées.

3.3.1.1 Données structurées

Des données structurées suivent un schéma prédéfini. Les données vont se conformer aux spécifications fournies dans le schéma. Comme pour le modèle relationnel, le modèle orienté graphe est conçu pour stocker des données structurées. Ces données sont stockées dans les nœuds du graphe.

3.3.1.2 Données faiblement structurées

Cela signifie qu'il n'y a aucune structure identifiable pour ces données. Les données non-structurées sont aussi décrites comme des données ne pouvant pas être stockées dans des tables, dans une base de données relationnelle (un document dans un répertoire, une vidéo, une image).

La part des données non-structurées est toujours plus importante au sein des entreprises. Ces données existent sous forme de texte stocké dans des documents, manuels, rapports, messages, pages Web et présentations. Certains SGBD relationnels (Oracle ou Postgres) prennent en charge ce type de données. Ces données faiblement structurées vont cohabiter à côté des données structurées. Une seule et même requête sera capable de travailler avec les deux types de données simultanément (recherche d'un terme dans un document et avec un filtre sur le nom de l'auteur).

Neo4J est un logiciel Open Source, qui implémente le modèle orienté graphe. Il ne possède pas de fonctionnalités pour stocker et travailler avec du texte libre de grande taille. Neo4J est répandu et bien documenté, de surcroît il a l'avantage d'être une base de données conforme aux propriétés ACID, qui stocke des données structurées sous la forme de graphe.

3.3.2 La relation entre les données

Dans le modèle orienté graphe, les données sont reliées entre elles localement par des relations. Si on dissocie tous les tuples pour un ensemble de tables et que l'on conserve toutes ces relations, alors on obtient un graphe. Ce concept est représenté dans la Figure 74 et la Figure 75. Le modèle orienté graphe stocke l'ensemble de ces relations. La flexibilité de ce modèle permet de rajouter des nœuds et des relations, sans compromettre les données existant. Ce type de modèle est parfaitement adapté pour les données connectées.

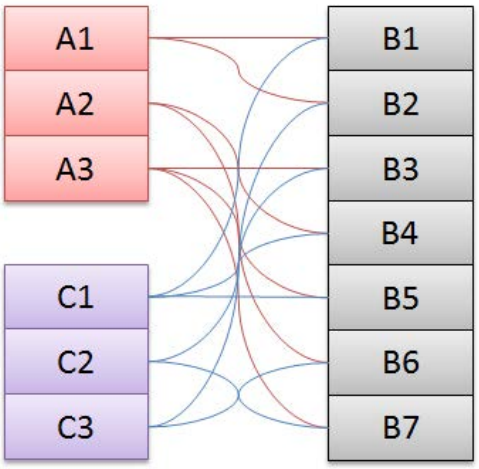

Figure 74 : Organisation des tuples dans un SGBD relationnel

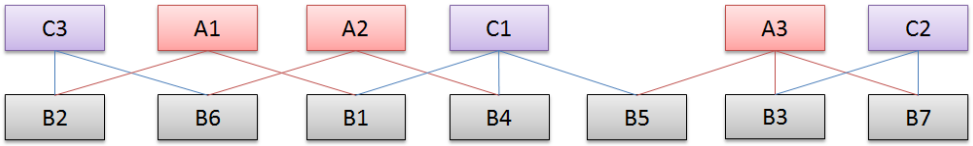

Figure 75 : Organisation dans un modèle orienté graphe

Les requêtes vont utiliser des chemins qui sont stockés de façon naturelle dans le graphe. Par conséquent, ces requêtes seront beaucoup plus performantes que dans d'autres modèles. Pour le cas du modèle relationnel, les requêtes impliquent souvent l'utilisation de jointures entre les tables composant le schéma. Ceci a des conséquences sur les performances du système.

Si on considère l'ensemble des modèles utilisés par les technologies NoSQL, c'est le modèle orienté graphe qui se distingue le plus du modèle relationnel. Dans le modèle relationnel, les tables contiennent des ensembles de lignes, elles-mêmes composées d'un ensemble de valeurs. Les relations correspondent aux jointures réalisées entre les lignes des différentes tables, elles ne correspondent pas à des relations de sémantique entre les termes (les valeurs). Dans le modèle orienté graphe, les valeurs ne font pas partie d'un ensemble, elles sont directement reliées entre elles.

3.3.2.1 L'intégrité référentielle

Les contraintes d'intégrité référentielles, permettent au SGBD de gérer automatiquement la présence de données référencées dans différentes relations de la base. La notion de lien permet de spécifier de telles propriétés.

La transformation d'un modèle conceptuel en un modèle physique est relativement intuitive, si on utilise des graphes. Cette transformation n'est pas toujours évidente lorsqu'on travaille dans le monde relationnel. En effet lorsqu'on passe d'un schéma entité-relation à un modèle logique de données, les contraintes d'intégrité référentielles sont ajoutées. Toutes ces métadonnées servent à la base de données mais pas directement à l'utilisateur. Les graphes maintiennent naturellement l'intégrité référentielle qui existe entre les données.

3.3.2.2 La relation hiérarchique

Les relations de hiérarchie sont représentées par des arbres. Les relations entre les nœuds de cet arbre, sont de type parent-enfant. Les graphes gèrent naturellement les relations hiérarchiques qui existent entre les données. Ces structures sont facilement exploitées à l'aide des algorithmes de parcours de graphe. En revanche, les traitements des données ayant des liens de parenté, sont difficiles à gérer dans une base de données relationnelle. Le problème vient du fait que le SQL standard ne permet pas le parcours récursif des données. Pour pallier ce problème, des fonctions spécifiques ont été ajoutées pour étendre le SQL standard et combler cette lacune, il s'agit par exemple de la fonction « START WITH / CONNECT BY » implémentée dans Oracle.

3.3.3 Le cycle de vie

La technique MVCC est une méthode de contrôle des accès concurrents fréquemment utilisés dans les systèmes de gestion des bases de données. Lorsqu'une donnée est modifiée dans la base, une nouvelle version est créée. Ainsi, les données anciennes sont conservées. Ce mécanisme permet d'éviter l'utilisation de verrous pour les opérations en lecture / écriture. Les données obsolètes devront être purgées par le système. De nombreuses bases de données utilisent le MVCC, comme MySQL, Postgres et Oracle. Ce mécanisme est également implémenté, dans certains SGBD NoSQL, tel que CouchDB.

3.3.3.1 La gestion des versions

La gestion des versions, dite Versioning, est généralement associée à une stratégie de migration de données, qui a pour but de déplacer les données d'une ancienne version vers une nouvelle. Dans le cas du modèle orienté graphe, la gestion des versions peut être conceptualisé à l'aide d'un graphe:

$$(V3) - [:PREV] -> (V2) - [:PREV] -> (V1)$$

Les nœuds du graphe comportent les numéros de version. Dans l'exemple cité, la version la plus récente est contenue dans un nœud qui comporte la propriété « V3 ». Ces numéros de versions sont reliés entre eux par les relations de type « previous ».

3.3.4 Le schéma et les opérations CRUD

Dans un SGBD, les tables sont définies par un schéma structurant les données stockées, sur lesquelles agissent les opérations CRUD.

3.3.4.1 Le schéma

On parle de tables dans le cas des bases de données relationnelles. Bien qu'il n'y ait pas à proprement parler de schéma dans le modèle orienté graphe, le typage (string, integer) des valeurs qui rentrent dans la composition d'une propriété (couple clé-valeur) est possible avec Neo4J. Il est possible également, de créer des contraintes. Celles-ci vont s'assurer que les valeurs des propriétés sont uniques pour un ensemble de nœuds. La commande suivant, va créer une contrainte d'unicité pour le numéro d'identification d'un article.

$$CREATE\ CONSTRAINT\ ON\ (article{:}\ Article)\ ASSERT\ article.id\ IS\ UNIQUE$$

3.3.4.2 Les opérations CRUD

Pour rechercher de l'information dans un modèle orienté graphe, la technique utilisée est appelée Graph Walking. Il existe plusieurs langages pour parcourir des graphes dans Neo4J, tels que Gremlin, le langage SPARQL et Cypher.

SPARQL est un langage de requêtes au même titre que le SQL pour les bases de données relationnelles qui permet de rechercher, d'ajouter, de modifier ou de supprimer des données RDF disponibles, à travers Internet. Son nom est un acronyme qui signifie SPARQL Protocol and RDF Query Language.

Cypher, spécifique à Neo4J, est choisi comme exemple car il est facile à comprendre et il suit une logique de représentation de graphes. Cypher permet d'effectuer des recherches sur des données qui sont en correspondance avec un motif spécifique. Il y a différentes clauses pour construire des requêtes, les 3 clauses principales sont « START », « MATCH » et « RETURN ». La clause « START » permet de spécifier un point de départ, la clause « MATCH » permet de fournir un motif spécifique et la clause « RETURN » renvoie à l'utilisateur les éléments pour lesquels il existe une correspondance avec le motif. Les commandes qui suivent vont créer le graphe représenté dans la Figure 73. Dans l'exemple suivant on crée l'utilisateur « Alice », 2 commandes (chacune avec un identifiant et 2 propriétés: « date » et « status ») et 3 produits (chacun avec un identifiant et une propriété: « description »).

$CREATE\ (\{\ user\colon 'Alice'\});$
$CREATE\ (\{\ order\colon 1234, date\colon 20120808, status\colon 'delivered'\ \});$
$CREATE\ (\{\ order\colon 5678, date\colon 20120816, status\colon 'dispatched'\ \});$
$CREATE\ (\{\ id\colon 'abcd', description\colon 'strawberry\ ice\ cream'\ \});$
$CREATE\ (\{\ id\colon 'efab', description\colon 'brussels\ sprouts'\ \});$
$CREATE\ (\{\ id\colon 'cdef', description\colon 'espresso\ beans'\ \});$

L'exemple suivant montre comment les commandes sont reliées avec les produits.

$MATCH\ (order\ \{order\colon 1234\}), (article\ \{id\ :\ 'abcd'\})$
$CREATE\ (order) - [\colon CONTAINS] -> (article);$

$MATCH\ (order\ \{order\colon 1234\}), (article\ \{id\ :\ 'efab'\})$
$CREATE\ (order) - [\colon CONTAINS] -> (article);$

$MATCH\ (order\ \{order\colon 1234\}), (article\ \{id\ :\ 'cdef'\})$
$CREATE\ (order) - [\colon CONTAINS] -> (article);$

$MATCH\ (order\ \{order\colon 5678\}), (article\ \{id\ :\ 'cdef'\})$
$CREATE\ (order) - [\colon CONTAINS] -> (article);$

Il reste à présent à relier chaque commande avec l'utilisateur Alice et à créer les relations pour pouvoir retrouver les commandes chronologiquement.

$MATCH\ (user\ \{user: 'Alice'\}), (order\ \{order: 5678\})$
$CREATE\ (user) - [: PLACED] -> (order);$
$MATCH\ (user\ \{user: 'Alice'\}), (order\ \{order: 1234\})$
$CREATE\ (user) - [: PLACED] -> (order);$

$MATCH\ (user\ \{user: 'Alice'\}), (order\ \{order: 5678\})$
$CREATE\ (user) - [: MOST\_RECENT] -> (order);$
$MATCH\ (o2\ \{order: 5678\}), (o1\ \{order: 1234\})$
$CREATE\ (o2) - [: PREVIOUS] -> (o1);$

La commande suivant retourne l'ensemble du graphe stocké dans la base de données :

$MATCH\ (n) - [r] -> (m)$
$RETURN\ n\ as\ from, r\ as\ `->`, m\ as\ to;$

Pour supprimer l'entrée complète du graphe, on peut utiliser cette commande :

$MATCH\ (n)$
$OPTIONAL$
$MATCH\ (n) - [r] - ()$
$DELETE\ n, r\ ;$

Les opérations qu'il est possible de réaliser sur les graphes sont très souples. Cependant, la création de nouveaux nœuds avec leur relation est peu performante.

3.3.5 La consistance des données

La consistance est le deuxième composant des quatre propriétés ACID.

3.3.5.1 Les propriétés ACID

Il s'agit de l'ensemble de propriétés, qui garantissent que les transactions dans la base de données sont fiables :

1- L'atomicité est l'opération exécutée sur des données, qui réussit entièrement ou sinon échoue entièrement.
2- La consistance, signifie qu'une transaction se termine normalement et après validation des résultats, la base de données est préservée dans un état consistant.
3- L'isolation, signifie que des évènements appartenant à une transaction doivent être invisibles pour d'autres transactions concurrentes.
4- La durabilité est la propriété qui assure, que lorsqu'une transaction est validée, elle est persistante dans le système, même si une panne survient après la validation.

Neo4J est conforme aux propriétés ACID. Cela signifie que les opérations sont exécutées dans le système en respectant les contraintes ACID. Le niveau d'isolation par défaut est « READ_COMMITED ». Ce niveau d'isolation garantit qu'une transaction va lire uniquement des données qui sont validées, dites Committed Values et éviter ainsi, les opérations de Dirty Read, lorsqu'une transaction lit une donnée modifiée par une autre transaction, sans que la modification ait été validée.

Neo4J possède également, un mécanisme de détection des verrous mortels, dit Dead Lock. Cela se produit lorsque deux transactions se bloquent mutuellement. La suppression d'un nœud ou d'une relation est effectuée de façon consistante, toutes les propriétés liées à ce nœud ou à cette relation seront également supprimées. Dans la version actuelle de Neo4J, il est possible de configurer des contraintes d'unicité pour certaines entités composant le graphe.

3.3.6 La performance

Les mécanismes d'exécution d'une requête sur le modèle orienté graphe et ceux d'une requête SQL sont assez différents. Par exemple, si on souhaite connaître les noms des agents qui suivent l'utilisateur numéro 23, une des solutions serait d'écrire la requête SQL suivante :

$$SELECT\ agents.*$$
$$FROM\ users$$
$$INNER\ JOIN\ users\_agents\ ON\ users.userid\ =\ users\_agents.userid$$
$$INNER\ JOIN\ agents\ ON\ users\_agents.agent\_id\ =\ agents.agent\_id$$
$$WHERE\ users.userid\ =\ 23;$$

Les trois piles de la Figure 76 représentent 3 tables: « users », « agents » et « users_agents ». La table « users_agents » est une table d'association, elle permet de relier les données des tables « users » et « agents ». La requête SQL va devoir aller chercher des données dans ces trois tables pour répondre à notre problème. La même requête est formalisée cette fois-ci avec le langage Cypher :

$$START\ user\ =\ node(23)$$
$$MATCH\ user - [user - agent] * agent$$
$$RETURN\ agent$$

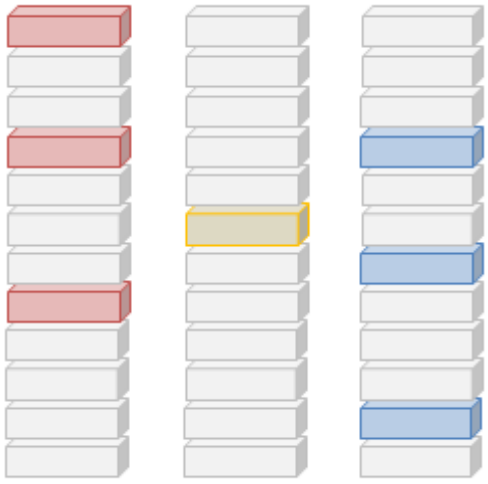

Figure 76 : Répartition des données dans le modèle relationnel

La Figure 77, met en évidence le nombre d'éléments impliqués dans la requête. La relation « [user-agent] » permet de faire directement le lien entre l'utilisateur et ses compétences. Il n'est pas nécessaire d'utiliser une table d'association comme dans le modèle relationnel. De plus, pour obtenir ce résultat, seule une petite portion du graphe est parcourue.

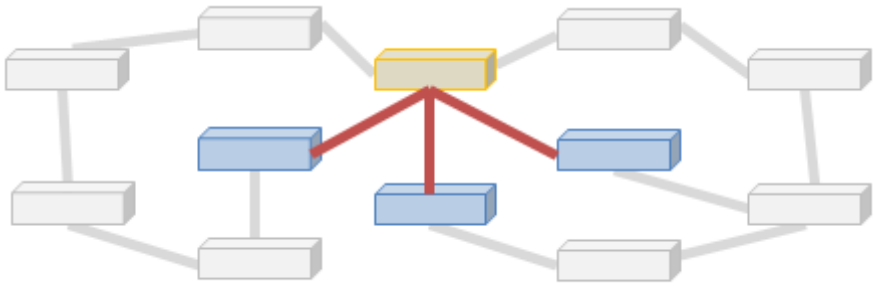

Figure 77 : Répartition des données dans le modèle orienté graphe

3.3.6.1 L'index

Les SGBD relationnels sont dotés depuis longtemps, de mécanismes d'indexation performants. L'utilisation de statistiques calculées à partir des données contenues dans les tables, permet d'optimiser l'utilisation de ces index. Neo4J offre la possibilité d'indexer les propriétés associées aux nœuds du graphe. Les index dans Neo4J sont de type « Eventuellement Disponible ». Cela signifie que la commande retourne immédiatement lors de la création d'un index. La création de l'index se fait en tâche de fond. Pendant la construction de l'index, les requêtes continuent de s'exécuter sur un graphe sans index. Les index sont maintenus automatiquement par le système.

3.3.6.2 La volumétrie

Dans le Tableau 19, on retrouve une comparaison des performances entre un SGBD relationnel et Neo4J pour un réseau social constitué avec un million d'utilisateurs, chacun en relation avec 40 amis, les relations entre les utilisateurs vont jusqu'à un niveau de profondeur 5, une étude menée par Partner et Vukotic en 2013.

Profondeur	Temps d'exécution SGBD relationnel	Temps d'exécution Neo4j	Nombre d'enregistrements concernés
2	0,016 s	0,01 s	~2500
3	30,267 s	0,168 s	~110000
4	1543,505 s	1,359 s	~600000
5	∞	2,132 s	~800000

Tableau 19 : Comparaison des performances entre un SGBD relationnel et Neo4J

A un niveau de profondeur 2, Neo4J et le SGBD relationnel montrent un temps de réponse similaire. Cependant, déjà à partir d'un niveau de profondeur 3, le SGBD relationnel donne un temps de réponse inacceptable pour une application Web en ligne. Le modèle orienté graphe est bien adapté pour des données connectées (c'est-à-dire des données qui sont liées sémantiquement entre elles). Chaque nœud possède des références directes par rapport aux nœuds qui lui sont adjacents. Cela signifie que le temps de réponse ne dépend pas de la taille globale du graphe mais seule une partie du graphe est concernée. La complexité de l'algorithme de recherche va être directement proportionnelle au nombre de nœuds traversés. Pour traverser un réseau de « n » nœuds, la complexité serait de « O(n) ».

Une base de données relationnelle est conçue pour fonctionner sur un serveur centralisé et garantir un certain nombre de propriétés liées aux transactions. Dans le cas où un système de base

de données nécessite plus de puissance (volume de données qui augmente, mise en production de nouvelles applications) il faudra migrer les données de la base, sur une machine de calibre supérieur de type Upgrade in the Box, ce qui en soi est une opération compliquée et qui nécessite un arrêt du service.

Comme toutes les bases NoSQL, Neo4J fait partie de la catégorie des bases de données BigData, les données peuvent être distribuées sur plusieurs machines afin de répartir la charge. La réplication entre un nœud maître et des nœuds esclaves permet également, d'augmenter les performances en diminuant le temps de réponse pour les requêtes en lecture.

3.3.7 L'analyse

Les rapports d'analyse ne sont pas générés directement sur les systèmes opérationnels. Les entrepôts de données stockent les données courantes et historiques, extraites de divers systèmes opérationnels et les regroupent dans le but de permettre la génération de rapports. Les données une fois entrées dans l'entrepôt sont stables, en lecture seule, non modifiables par les utilisateurs. Ceci afin de conserver la traçabilité des informations et des décisions prises. Les DWH possèdent une structure particulière, permettant de stocker des données qui sont dé-normalisées. Il s'agit d'un modèle en étoile qui s'écarte du modèle relationnel.

3.3.7.1 Les rapports et la prédiction

Généralement il y a de très gros volumes de données au sein d'un entrepôt, c'est un modèle performant pour faire de l'analyse. Ce modèle qui fonctionne aussi à l'intérieur d'un SGBD relationnel, vient compléter le modèle relationnel. Hormis les réseaux sociaux, il y a beaucoup de domaines pour lesquels il est intéressant de représenter les données à l'aide d'un graphe, par exemple la bio-informatique, les statistiques en sport ou le commerce électronique. Ces secteurs d'activité vont également bénéficier, de ce modèle pour faire de l'analyse. On reprend comme référence l'exemple de la Figure 75 sur laquelle sont représentées les commandes. Il est possible de construire un petit moteur d'inférence extrêmement simplifié pour réaliser de la vente croisée. On a remarqué que l'utilisateur « Alice » commande fréquemment les produits « abcd » et « cdef », l'objectif est de recommander un de ces produits à un autre utilisateur qui possède certains points en commun avec « Alice ». On peut rechercher un utilisateur qui possède un certain nombre de points communs avec « Alice », par exemple un utilisateur qui a son lieu de résidence proche de chez « Alice » et qui a commandé le produit « cdef » mais pas le produit « abcd ». Cet utilisateur peut potentiellement s'intéresser au produit « abcd ».

La requête suivant va permettre d'obtenir ce résultat :

$$MATCH\ (article) <-[:CONTAINS]-(order) <-[:PLACED]-(user)-[:NEAR]->(uAlice\ \{user:Alice\})$$
$$WHERE\ article.id\ =\ 'cdef'\ AND\ article.id\ <>\ 'abcd'$$
$$RETURN\ user;$$

3.4 Conclusion

Le Tableau 20 est une synthèse de l'ensemble des critères évoqués, pour les deux modèles étudiés précédemment. La couleur rouge signifie un mauvais état, une dégradation des performances ou un critère non implémenté. La couleur jaune signifie une réalisation possible et la couleur verte met en avant les bonnes performances d'un critère ou un critère clé du modèle.

Rubriques	Catégories	Modèle orienté graphe	Modèle relationnel
Données	Nature des données	Données structurées	Données structurées
		Données faiblement structurées	Données faiblement structurées
	Relation entre les données	Intégrité référentielle	Intégrité référentielle
		Relations hiérarchiques	Relations hiérarchiques
	Cycle de vie	Gestion des versions	Gestion des versions
Requêtes	Schéma et opérations CRUD	Schéma	Schéma
		Opérations CRUD	Opérations CRUD
	Consistance des données	Propriétés ACID	Propriétés ACID
Performance	Performance	Index	Index
		Volumétrie	Volumétrie
Analyse	Analyse	Rapports et prédiction	Rapports et prédiction

Tableau 20 : Critères de performance du modèle orienté graphe et du modèle relationnel

Bien que le modèle orienté graphe soit une très bonne solution dans un contexte BigData, il peut être également, une solution dans d'autres situations, pour lesquelles la gestion d'un grand volume des données n'est pas la préoccupation centrale. C'est un modèle évolutif, moins rigide que le modèle relationnel. Il est mieux adapté pour supporter les modifications qu'impose l'évolution du marché. C'est un modèle de choix pour traiter des données connectées entre elles ou faire de l'analyse. Pour l'aspect transactionnel, le modèle orienté graphe est conforme aux propriétés ACID. Cependant, si on se place dans un contexte dans lequel les données sont répliquées sur plusieurs nœuds, le système va perdre en consistance. On dit alors qu'il est de type Eventually Consistant. D'un point de vue de la sécurité des données, Neo4J n'offre pas des mécanismes aussi performants que ceux que l'on trouve au niveau de certains SGBD relationnels (Oracle, SQL Server) qui implémentent une très grande granularité dans la configuration des droits d'accès aux données.

Un seul modèle est rarement optimal pour répondre à l'ensemble des besoins d'une application et le modèle orienté graphe avec ses caractéristiques est un bon complément au modèle relationnel.

Annexe 4. Les publications dans des conférences et des journaux internationaux

4.1 Introduction

Durant l'élaboration de ce travail, on a participé à un ensemble de conférences très actives dans ce domaine de recherche afin de publier, partager et échanger de nouvelles idées et de nouvelles approches sur la problématique de la modélisation des données BigData. Ainsi, on a présenté ces recherches dans des conférences spécialisées sous forme d'articles présentés à une audience de professionnels. Ces discussions ont permis de développer ce travail et de faire aboutir ce projet de l'approche de la modélisation intégratrice en général ou via le raisonnement par étude de cas en particulier.

Dans cette annexe, on va d'abord présenter les publications par ordre chronologique. Ensuite, on va présenter le développement de notre approche à travers ces publications.

4.2 Les publications par ordre chronologique

Ce travail de recherche sur la modélisation des données BigData, commence en 2013 et dure jusqu'à 2016 pour cadrer ce travail de thèse de doctorat.

Tout d'abord, concernant les articles publiés ou acceptés pour publication dans des conférences internationales :

[1] H. Hashem et D. Ranc, An Integrative Modeling of BigData Processing, 9th International Conference on Intelligent Systems: Theories and Applications (SITA), 2014 à Rabat, prix du meilleur article.

SITA 2014.pdf

[2] H. Hashem et D. Ranc, Predicate-based Cloud Computing, 8th International Conference on Next Generation Mobile Apps, Services and Technologies (IEEE NGMAST), 2014 à Oxford, Print ISBN 978-1-4799-5072-0.

IEEE NGMAST 2014.pdf

[3] H. Hashem et D. Ranc, Extending Standard MapReduce Algorithms, 2nd International Conference on Multimedia Big Data (IEEE BigMM), 2016 à Taipei, print ISBN 978-1-5090-2179-6.

IEEE BigMM 2016.pdf

[4] H. Hashem et D. Ranc, A Review of Modeling Toolbox for BigData, 15th International Conference on Military Communications and Information Systems (IEEE ICMCIS - former MCC), 2016 à Bruxelles, Print ISBN 978-1-5090-0472-0.

IEEE ICMCIS 2016.pdf

[5] H. Hashem et D. Ranc, Data Modeling and Case-based Reasoning for Social Monitoring, International Symposium on Advanced Big Data and Applications (IEEE ABA), 2016 à Vienne, print ISBN 978-1-5090-3946-3.

IEEE ABA 2016.pdf

[6] H. Hashem et D. Ranc, Evaluating NoSQL document oriented data model, 3rd International Symposium on Intercloud and IoT (IEEE ICI), 2016 à Vienne, print ISBN 978-1-5090-3946-3.

IEEE ICI 2016.pdf

Ensuite les articles publiés ou acceptés pour publications dans des journaux internationaux :

[7] H. Hashem et D. Ranc, An Integrative Modeling of BigData Processing, International Journal of Computer Science and Applications (IJCSA), Technomathematics Research Foundation, Vol. 12, No. 1, pp. 1 – 15, 2015 à Kolhapur, ISSN 0972-9038.

IJCSA 2015.pdf

[8] H. Hashem et D. Ranc, Pre-processing and Modeling Tools for BigData, Foundations of Computing and Decision Sciences Journal (FCDS), Vol. 41, No. 3, pp. 151 – 162, 2016 à Berlin, ISSN 0867-6356.

FCDS 2016.pdf

4.3 Le développement des publications

La première année, ce travail était focalisé sur la nature des bases de données non-relationnelles et les différents modèles de données et techniques de modélisation disponibles afin de préparer la conception de l'idée de modélisation intégratrice. Ainsi, dans la publication « An Integrative Modeling of BigData Processing », on présente ces aspects théoriques, en illustrant les propos par une expérience agile menée sur des données Twitter. Le prix du meilleur article de la conférence a été accordé à ce papier [1], qui a été publié par la suite dans un journal international [7].

Ensuite, on a orienté la recherche dans le domaine des systèmes experts et le raisonnement par étude de cas. Ainsi, on a défini un concept sous forme de râteau, permettant de prétraiter les données, dont les dents représentent les quatre étapes du cycle de raisonnement par étude de cas, présentées dans la publication « Predicate-based Cloud Computing ». Cette publication est apparue dans la librairie digitale IEEE [2].

Dans la deuxième moitié de la deuxième année, on a travaillé pour consolider les notions théoriques afin d'élaborer un concept général de la modélisation BigData, permettant d'étendre le périmètre d'utilisation du Framework MapReduce. C'était le sujet des 2 publications, « Extending Standard MapReduce Algorithms » et « A Review of Modeling Toolbox for BigData », dont l'apparition dans la librairie digitale IEEE est prévue [3] [4]. Une version étendue du papier « Pre-processing and Modeling Tools for BigData », est également publiée dans un journal international [8].

On a appliqué par la suite le concept de pré-traitement avec un raisonnement par étude de cas dans le contexte des réseaux sociaux. C'était le sujet traité par la publication « Data Modeling and Case-based Reasoning for Social Monitoring », dont l'apparition dans la librairie digitale IEEE est prévue [5].

En dernière année, on a publié une étude sur les bases de données orientées document. Il s'agit de la publication « Evaluating NoSQL document oriented data model » acceptées pour publication dans la librairie digitale IEEE également [6].

4.4 Conclusion

Dans cette annexe, on a présenté les articles et les publications réalisés dans des conférences internationales pour faire connaitre ce travail et aussi pour partager et échanger avec les professionnels du domaine. D'abord, on a présenté les publications par ordre chronologique. Ensuite, on a présenté le développement de ces publications selon l'approche de la modélisation intégratrice afin de répondre à la problématique du travail. Enfin, on a listé également les publications dans des journaux internationaux.